VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Metriky a měření kvality softwarového vývoje

Metrics and Software Quality Measurement

2012 Bc. Aleš Kaňák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student:	**Bc. Aleš Kaňák**
Studijní program:	N2647 Informační a komunikační technologie
Studijní obor:	2612T025 Informatika a výpočetní technika
Téma:	Metriky a měření kvality softwarového vývoje Metrics and Software Quality Measurement

Zásady pro vypracování:

Cílem této diplomové práce je vypracovat aplikační framework pro metriky a měření kvality softwarového vývoje. Měření a s tím spojené zkoumání kvality prochází celým softwarovým vývojem od zadání projektu až po jeho ukončení a vyhodnocení. Framework pro metriky a měření kvality by měl postihnout celý tento proces. Práce je určena pro více studentů, kteří budou spolupracovat na tvorbě celého frameworku.

Framework bude obsahovat zejména tyto oblasti:
1. Zaměření se na možnosti odhadu velikosti a potřebného úsilí na projekt a jejich kontrola v průběhu projektu.
2. Metriky a postupy pro zajištění kvality vývoje , detekci chyb a vylepšování procesů.
3. Aplikační framework pro implementaci všech navržených metrik, měření a vyhodnocení, metodiku pro zpracování dat.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Svatopluk Štolfa, Ph.D.**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 4. května 2012

Děkuji Ing. Svatopluku Štolfovi, Ph.D. za vedení a odbornou pomoc při zpracování a tvorbě diplomové práce. Taktéž děkuji svým kolegům – Bc. Jakubu Kamrlovi a Bc. Pavlu Benešovi, se kterými jsem práci společně vytvářel, za zodpovědný přístup a týmovou spolupráci.

Abstrakt

Tato diplomová práce se zabývá studiem problematiky metrik a vytvořením aplikačního frameworku, který umožní zkoumat kvalitu softwarového vývoje. První část je věnována studiu problematiky metrik. Jedná se jak o studium odborných textů, tak o průzkum a analýzu stávajících řešení. Druhá část práce se zabývá analýzou a implementací aplikačního frameworku pro sledování kvality vývoje softwaru.

Klíčová slova

.NET, agilní metody, aplikační framework, C#, JIRA, metriky, měření kvality, odhady velikosti a vynaloženého úsilí, Scrum, Team Foundation Server, vývoj softwaru, webová aplikace

Abstract

This thesis studies the metrics and the creation of the application framework, which in turn will examine the quality of the software development. The first section is devoted to the study of the metrics involved. It is a study of the scientific texts and a survey and analysis of existing solutions. The second section deals with the analysis and implementation of the application framework for monitoring the quality of software development.

Keywords

.NET, agile methods, application framework, C#, JIRA, metrics, quality measurement, size and effort estimation, Scrum, Team Foundation Server, software development, web application

Seznam použitých symbolů a zkratek

CSV	Comma-separated values
FP	Functional Point
FSM	Functional Size Measurement
GUI	Graphical User Interface
HTTPS	Hypertext Transfer Protocol Secure
IID	Incremental and Iterative Development
ISO	International Organization for Standardization
ITIL	Information Technology Infrastructure Library
LLOC	Logical Lines of Code
LOC	Lines of Code
MS	Microsoft
OS	Operating System
SQL	Structured Query Language
TFS	Team Foundation Server
URL	Uniform Resource Locator
XML	Extensible Markup Language

Obsah

1 Úvod

V dnešní době často neřešíme, jak přístroje pracují nebo jak byly vytvořeny, ale pouze využíváme jejich funkce. Víme například, že mobilní telefon nám zjednodušuje komunikaci mezi lidmi, umožňuje nám volat, psát, připojit se na internet, apod. Zapomínáme ale na fakt, že se skládá z mnoha částí. Proces výroby takového výrobku je velmi složitý a skládá se z mnoha etap. Nejinak je tomu při tvorbě softwarového díla. Každý projekt je tvořen jeho realizací a řízením. Před samotnou implementací je provedena jeho detailní analýza a návrh. Než se přístroj uvede do ostrého provozu, předchází mu fáze testování, při které se snažíme snížit defekty na požadovanou mez. Jak to v životě bývá, tak ne vždy jde vše podle plánu. Tyto a mnohé další skutečnosti spravuje projektové řízení, které se skládá z plánování a řízení. Jeho funkce, zjednodušeně řečeno, spočívá v dodání kvalitního produktu, pokud možno včas a s co nejlepším finančním ziskem. Bez kontroly procesů by mohl být výsledek práce katastrofální – například mobilní telefon, se kterým by nešlo telefonovat.

Cílem této diplomové práce je vypracovat *aplikační framework* pro metriky a měření kvality softwarového vývoje. Měření a s tím spojené zkoumání kvality prochází celým softwarovým vývojem od zadání projektu až po jeho ukončení a vyhodnocení. Framework pro metriky a měření kvality postihuje celý tento proces. Na práci jsem spolupracoval s dalšími studeny a společně jsme se tak podíleli na tvorbě celého frameworku.

Diplomová práce je rozdělena do dvou tematických celků. První část, která obsahuje studium problematiky metrik, je popsána ve druhé až sedmé kapitole. Druhá část se zabývá vývojem *aplikačního frameworku* a najdeme ji v osmé až desáté kapitole.

1.1 Studium problematiky

Obsahem teoretické části je jednak seznámení se základní problematikou metrik, včetně jejich metod, tak i seznámení se s moderními přístupy vývoje softwaru a nástroji dostupnými na trhu.

Nejprve nastíním naši motivaci a důvody, proč je tato problematika zajímavá a hlavně velice užitečná při tvorbě nejenom softwarového díla. V další kapitole následuje teorie metrik, kde zmíním jejich funkce a také základní metody, které se běžně používají. Zaměřím se zde na vynaložené úsilí a odhady velikosti projektu. V páté kapitole volně přejdu k metodám vývoje softwaru, které právě metriky hojně využívají. Zaměřím se především na agilní metody, ve kterých se budu věnovat i jejich historii.

Šestá kapitola je věnována přehledu dostupných nástrojů, které dokáží měřit kvalitu softwaru. Zaměřím se zde hlavně na nástroje používající *Scrum*. Další kapitola obsahuje podrobnou analýzu metrik jednoho vybraného nástroje, u kterého jsme čerpali nápady a inspiraci při tvorbě našich metrik.

1.2 Analýza a realizace aplikačního frameworku

Druhý tematický celek je věnován vývoji *aplikačního frameworku* pro agregaci dat a vyhodnocení kvality vývoje softwaru, přičemž osmá kapitola v něm popisuje analýzu a návrh. Řeší se zde vše od základních požadavků na funkcionalitu, struktura aplikace pro schopnost importu dat z ostatních systémů apod. Ještě před samotnou implementací je samostatná kapitola věnována návrhu metrik pro naši aplikaci. Jednotlivé metriky jsou zde i názorně vysvětleny. Předposlední kapitola se už zabývá implementací aplikace. Je zde vysvětlena jak architektura, tak i určité popisy modulů. Nechybí ani vysvětlení základní myšlenky při tvorbě metrik a jejich logiky nebo ukázky aplikace samotné.

2 Motivace

V dnešním moderním světě se stále častěji setkáváme s pojmem proces. Jako proces označujeme určitou množinu kroků, která vede k vytvoření komplexního díla nebo vylepšení. Tyto kroky nemusí nutně následovat jeden po druhém, kde následující krok vyplývá z předchozího, ale můžeme jednotlivé činnosti paralelizovat. Jedním z důvodů, proč vytváříme typizované postupy je možnost opakovat je v budoucnu. Tento fenomén doby se vyskytuje snad ve všech oblastech lidské činnosti. V našem případě se budeme soustředit především na problematiku softwarového procesu a procesního managementu.

Základem každé správně fungující organizace je její správné řízení a řízení vnitřních procesů. Na toto téma bylo již sepsáno nespočet postupů a rad. Jedná se o různé typy norem, například ISO. V oblasti informačních technologií pak mluvíme především o seznamu doporučení, konceptů a postupů označovaných jako ITIL. Mnohdy je však těžké se ve změti těchto norem vyznat. S tím souvisí i značně abstraktní forma, kterou jsou tyto normy a doporučené postupy psány.

Každá organizace je však jiná a dané postupy se dají uplatnit jen do určité míry. Do jaké míry se jimi pak organizace řídí, záleží na přístupu vedoucích pracovníků a jejich zkušenostech. K tomu, aby vedoucí mohli správně a pružně zasahovat do řízení procesu, musí mít ucelené informace o průběhu procesu.

Z toho vyplývá potřeba tyto informace o procesu průběžně měřit a zaznamenávat. Abychom z takto shromážděných dat mohli něco zjistit, musíme navrhnout určitou množinu metrik. Tyto metriky by nám měly poskytnout vyšší úroveň pohledu na řízený proces. Při bližším zkoumání problematiky zjistíme, že faktorů, které potřebujeme během každého procesu sledovat, je poměrně mnoho. Je tedy jasné, že systémy, které nám s řízením procesů pomáhají, jsou velmi rozsáhlé.

Častěji než v minulosti se při každé činnosti skloňuje pojem „úspora nákladů“. Konkurence na trhu je ve všech odvětvích vysoká. Cena požadovaného řešení se tak stává pro většinu společností klíčovou. Naším cílem je tedy snaha minimalizovat naše náklady na vývoj a nasazení. Další postupy společností při rozhodovacích procesech vyplývají z předešlých referencí nebo předešlé spolupráce. Ne vždy totiž jde cena ruku v ruce s kvalitou.

Při studiu této rozsáhlé problematiky bylo zjištěno, že v současnosti existuje na trhu široká škála nástrojů, které se zabývají sledováním procesů. Mezi těmito nástroji se dají nalézt menší či větší rozdíly. Jedná se především o to, jak velkou část životního cyklu procesu zpracovávají. Nepsaným pravidlem bývá fakt, že komerční řešení bývají ve většině případů vyspělejší. Často se liší po stránce množství a důležitosti vytvořených metrik, grafickým zpracováním či přívětivostí uživatelského rozhraní pro ovládání.

V praxi je zcela běžné, že zkušení vedoucí pracují externě ve více organizacích, které využívají různé systémy pro řízení. Z tohoto důvodu jsme se rozhodli vytvořit *aplikační framework* pro agregaci dat z více systémů a tím usnadnit celou agendu sledování a řízení. Další motivací bylo vytvořit nové a efektivnější metriky pro sledování procesů na základě shromážděných dat.

3 Teorie metrik

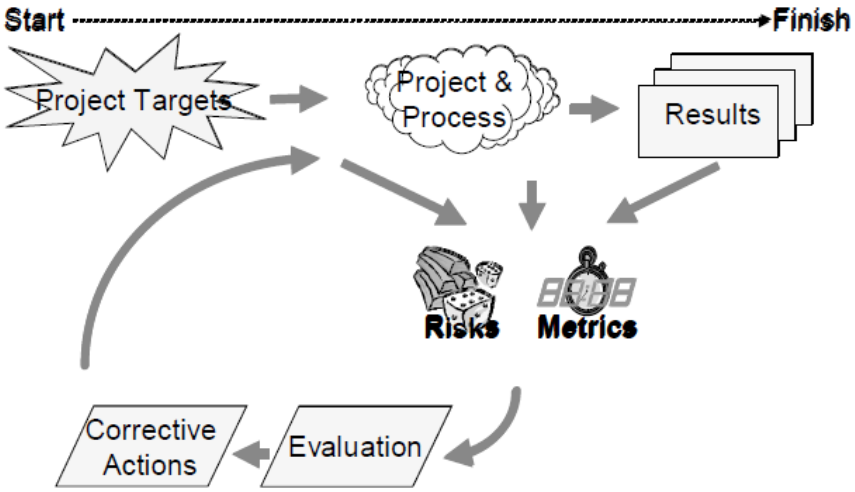
Abychom byli schopni vytvořit *aplikační framework* pro metriky a měření kvality softwarového vývoje a dokázali tak vyhodnocovat data, museli jsme se nejprve ponořit do teorie a dostat tak lepší povědomí o aktuální situaci.

Pro lepší pochopení a proniknutí do problematiky jistě pomůže následující konstatování: Pokud se chceme zlepšit, musíme si stanovit určitý cíl a měřit naše snažení. Naproti tomu, pokud si nestanovíme určitý cíl a nebudeme měřit naše snažení, potom neexistuje způsob, jak zjistit, zda jsme se zlepšili.

3.1 Projektové řízení a měření softwaru

Během života řešíme nespočet úkolů a každý z nás je řeší odlišným způsobem. Vynikáme tak v něčem více, v něčem naopak méně. Tyto aktivity mají většinou předem stanovený cíl, začátek a konec. Ne vždy je koordinace těchto aktivit jednoduchá, ale je tomu právě naopak. Naprosto stejné je tomu u projektů.

Nejprve se pojem projekt vyskytl ve stavebnictví, kde bylo zapotřebí koordinovat a plánovat činnosti více lidí. V managementu je často definován jako časově ohraničené úsilí, směřující k vytvoření unikátního produktu nebo služby. Kromě časového ohraničení a cíle je projekt vymezen svými zdroji. Pod tímto pojmem si můžeme představit lidské, materiální a finanční zdroje. Pro účely plánování a realizaci komplexních akcí, které je zapotřebí provést s plánovanými náklady a v čase, vzniklo projektové řízení. Je to proces, který identifikuje, měří, shromažďuje, plánuje a sleduje aktivity, rozhodnutí a náklady.



Obrázek 1: Projektové řízení [1]

Obrázek znázorňuje životní cyklus projektů a procesů (obrázek 1). Velmi důležité je na úplném začátku stanovit jasné a hlavně reálné cíle, které nám definují, čím se přesně bude daný projekt nebo

proces zabývat. Bylo by značně naivní si myslet, že se k výsledku dostaneme jednoduše a bez překážek. S tím částečně souvisejí rizika, která vznikají společně s projekty, kterých je celá řada. Může se jednat namátkou o špatnou motivaci pracovníků, neustálé měnící se požadavky, chyby při návrhu apod. Je jasné, že všem chybám se nemůžeme vyvarovat, ale na druhou stranu dobrým vedením je můžeme eliminovat na minimální úroveň. Velkou mírou nám v tomto ohledu pomáhají metriky, které ověřují a vyhodnocují naše dosažené výsledky a můžeme tak vytvořit korektivní opatření. Zmenšují míru překvapení, a to díky lepšímu přehledu o projektu. Nutno zdůraznit, že tento náhled dostáváme v průběhu projektu a ne pouze na jeho konci.

Dovolil bych si demonstrovat koloběh z obrázku 1 na ukázkovém příkladu. Máme sestrojit automobil, který má následující požadované parametry, především provozní a jízdní vlastnosti: obsah motoru 1,2 L, kombinovaná spotřeba 6L/100 km, akcelerace z 0 na 100 km/h za 12 s. Kdyby se jednalo o skutečný projekt, tak by zcela jistě měl daleko více cílů a požadavků, ale pro naše účely to bude zcela postačovat. Máme tedy definované cíle, a protože víme, co máme vytvářet, tak se pustíme do výroby vozu, dle dostupných informací a podkladů. Výsledkem bude hotový vůz, ale zatím vůbec nemáme tušení, zda jsme splnili počáteční požadavky. Bez ověření, otestování nebo proměření požadovaných vlastností by to bylo pouhé hádání. V této (a nejenom této) části přijdou na řadu metriky, díky nimž ověříme, zda jsou parametry takové, jaké jsme očekávali a naopak. Změříme tedy postupně všechny vlastnosti a můžeme je porovnat s požadovanými. Evidentně potom vidíme, pokud nastal v nějaké části problém a můžeme tak učinit rozhodnutí. Například pokud bude kombinovaná spotřeba vyšší, tak se budeme věnovat úpravám a optimalizací řídící jednotky, palivového potrubí atd. Poté opět provedeme kontrolní měření a budeme ho opakovat, dokud nebudeme mít automobil s požadovanými parametry a kýžené kvality. Tímto procesem snížíme možná rizika. Pokud by automobil měl například větší spotřebu, tak by nemusel obstát v konkurenci apod. Nicméně některým rizikům se vyhnout nemůžeme. Jedná se hlavně o ty externí, do kterých spadá například inflace, politické změny atd. Při výstupní kontrole všech automobilů, které opustí výrobní linku, by se rovněž dalo využít metrik, ale to se pořád bavíme spíše o kontrole, která je až na konci životního cyklu.

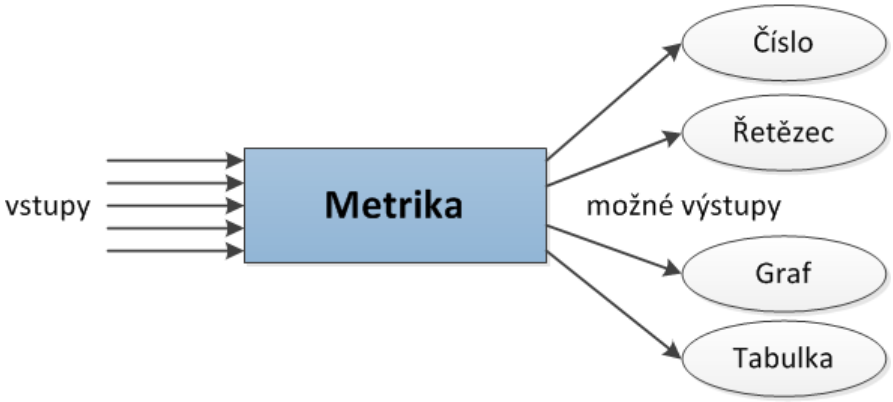
Největší síla a kouzlo metrik spočívá v informaci i během tohoto cyklu, ale ne vždy je toto možné. Závisí to hlavně na dostupných informacích a datech. V další podkapitole rozeberu problematiku metrik více dopodrobna.

3.2 Metrika

Nemůžeme kontrolovat něco, co nemůžeme změřit. Pokračoval bych tak zavedením pojmu *metrika*. Najdeme jej v oblastech matematické analýzy a teorii metrických prostorů, aplikovaných nejčastěji jako metrický tenzor nebo metriku sítě. Nás ale zajímají hlavně metriky procesů a projektů spadající do procesního a projektového řízení, nesoucí převážně kvalitativní charakteristiku. Budeme se s nimi dále setkávat v průběhu celé diplomové práce. Zcela záměrně jsem přímo nespojoval pojem metrika se softwarem v jakékoliv jeho formě, protože ne vždy musí být spojována právě s touto oblastí. Tento fakt vyplývá z historie, protože první použití metrik se datuje zhruba do doby, kdy ještě nebyly známy osobní počítače, natož software samotný. V našem případě budeme uvažovat pouze s metrikami, které se vážou na software. Nicméně základní princip je vždy stejný.

Vstupem každé metriky (obrázek 2) jsou dobře selektovaná data, která jsou sbírána v určitých časových intervalech a hlavně předchozí zkušenost v podobné situaci. Na základě algoritmu se tato

data vyhodnotí a pokud možno se nám zobrazí výsledek v číselné, textové nebo grafické formě. Díky tomu mohou vyplynout na povrch nové skutečnosti a získáme tak lepší povědomí a přehled o sledovaných položkách.



Obrázek 2: Metrika

Důvodů pro zavedení metrik je hned několik. Mohou nám pomoci ohodnotit produkty nebo procesy oproti stávajícím zavedeným standardům. Dále jsou užitečné při kontrole kvality produktu/projektu, nebo odhadu produktivity. Při plánování projektu mohou pomoci s odhadem nákladů a vynaloženého úsilí. Vhodnou aplikací zdokonalují procesy a zefektivňují tím výkonnost celé organizace a zvyšují její produktivitu.

Metriky vycházejí hlavně z předchozích zkušeností a porovnávají podobné procesy a projekty mezi sebou. Odměnou za kvalitně navrženou a vhodně aplikovanou metrikou může být například ušetřený čas, finance, ale také i cenné zkušenosti. Tyto vědomosti tak můžeme využít v budoucnu a dostaneme tak lepší odhad a hlavně kontrolu nad procesy či projekty.

Nastínil jsem, k čemu jsou metriky dobré a jaké jsou jejich benefity. Nicméně vytvořit dobrou metriku není vůbec jednoduché a tato situace se odráží i na trhu. Každá firma si pečlivě střeží své know-how ve svých řadách a pokud se o nějakou metriku nebo dokonce sadu metrik chce podělit, tak si za to nechá náležitě zaplatit. I přes tento fakt se vyplatí do metrik investovat nemalé prostředky, protože se to ve výsledku i přes zvýšenou režii vyplatí. V globálním měřítku je totiž pouze čtvrtina projektů dokončena v čase a s plánovaným rozpočtem. Zhruba 30 % je jich zrušeno před dodáním. Zbývající část je dokončena později, případně překročí plánovaný rozpočet anebo obsahuje pouze zlomek plánované funkcionality [1]. V dalších kapitolách se seznámíme s námi vytvořenými metrikami, které si popíšeme dopodrobna a ukážeme si, jak fungují v praxi.

3.3 Základní dělení metrik podle typu dat

Protože metriky odrážejí realitu v mnoha aspektech, tak se i dají dělit více způsoby. V následující podkapitole se tak budu snažit více přiblížit jejich nejběžnější rozdělení.

Typy metrik:

- Přímé
- Nepřímé/odvozené
- Predikce

První z nich je *přímé měření*. Tím je myšleno, že výsledkem je číslo, případně řetězec bez nějakého dalšího porovnání nebo odvození. Spadá zde např. počet řádků zdrojového kódu. Dalším typem jsou *nepřímé/odvozené metriky*. Může se jednat například o poměr úspěšných testovacích případů k jejich celkovému počtu. Další variantou metrik mohou být *predikce*, kdy na základně historických dat předpovídáme výsledek a porovnáváme ho s aktuálními daty.

Typy metrik:

- Nominální
- Ordinální
- Intervalové
- Poměrové

Následující dělení metrik se vztahuje hlavně na typ měřené hodnoty a její následnou interpretaci. Popisné *nominální metriky* slouží většinou k pouhému štítkování, kde se uvádí například informace o verzi, programovacím jazyku nebo poznámka. Pokud chceme řadit data do kategorií, tak můžeme využít *ordinálních metrik*. Značí často prioritu úkolů (malá, střední, vysoká, apod.) nebo určitou číselnou hodnotu a datum. Může se také opět jednat třeba o počet řádků zdrojového kódu, počet chyb, nebo strávený čas. *Intervalové metriky*, jak už název napovídá, určují, zda měřená hodnota spadá do rozmezí určitého intervalu. *Poměrové metriky* naopak udávají poměr mezi předpovídanou hodnotou a tou aktuální (např. počet chyb se za poslední měsíc zvedl 1,2×).

Typy metrik:

- Produktové
  - Velikosti
  - Složitosti
  - Kvality
- Procesní
- Zdrojů

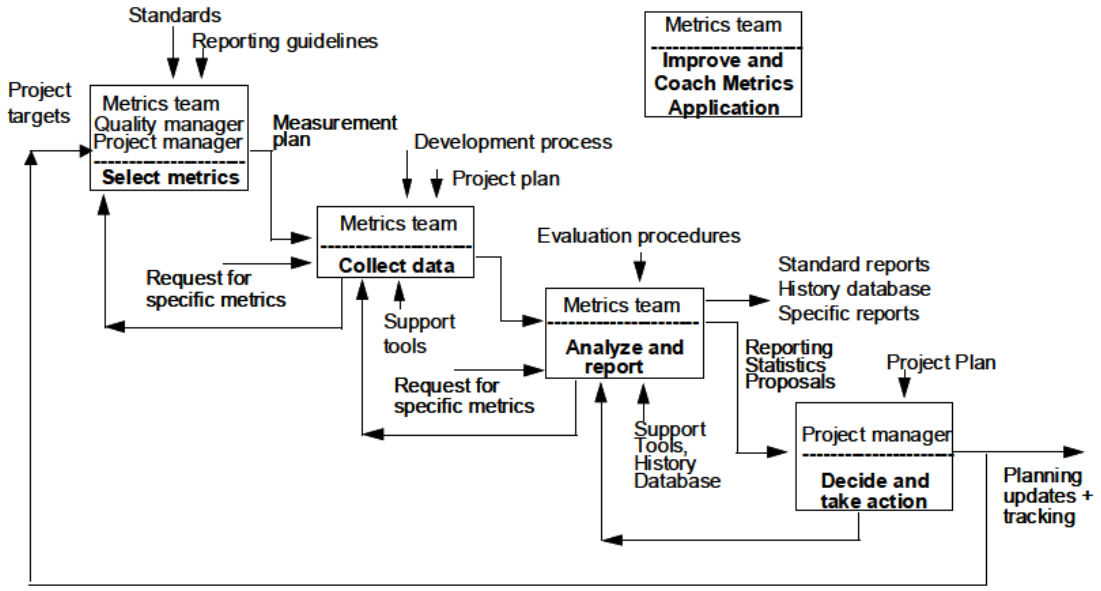
Produktové metriky se ještě dále dělí na ty, které se zabývají velikostí, složitostí a kvalitou. *Velikost* bych vysvětlil podrobně v další podkapitole a přešel tak k metrikám, které měří *složitost*. V této oblasti se často využívá teorie grafů. Co se týče *metrik kvality*, tak ty sledují nejčastěji počet chyb, počet úspěšných testovacích případů, počet nových vylepšení apod. Kategorie *procesních metrik* řeší převážně efektivitu managementu a měří výkony týmu vývojářů. Pokud potřebujeme sledovat vynaložené úsilí, tak využijeme *metrik zdrojů*. U těchto se kromě výše zmíněné skutečnosti ještě měří celkový čas a volné prostředky, ať už se jedná o pracovníky, nebo o techniku.

3.4 Zavedení metrik pro kontrolu nad projektem

Za selhání projektů nemůže ani tak neschopnost manažerů nebo nedostatečná technologie, nýbrž použití špatných manažerských technik. Výsledkem je pozdní vydání produktu, malá kvalita a daleko větší náklady, než s nimi bylo počítáno.

Kontrola projektu je klasický kontrolní proces, který je vidět v mnoha kontrolních systémech. Nejdůležitější je zde existence uzavřené smyčky mezi kontrolovaným objektem, metrikou aktuálního výkonu a srovnáním cílů s dosavadním průběhem. Měřící proces samotný se skládá ze čtyř základních částí: *výběr metrik*, *sběr dat*, *analýza a report* a *rozhodnutí včetně přijetí opatření* (obrázek 3).

O správný výběr metrik se stará ve větších organizacích samostatný tým věnující se metrikám, kterému dává impulsy i projektový manažer a manažer kvality. Na základě cílů projektu, aktuálních standardů a dostupných metod se vyselektují sady metrik. Získáme tak tímto procesem plán pro měření, díky kterému budeme vědět, jaká data bude tým věnující se metrikám sbírat během procesu vývoje, který probíhá dle projektového plánu. Ke sběru dat využijeme speciálních nástrojů a technik k tomu určeným. V další fázi probíhá vyhodnocení dat. Využijeme k tomu opět nástroje, kterými už budeme specifikovat požadavky na konkrétní metriky. Nicméně se vřele doporučuje použít i historická data, pokud jsou k dispozici. Nejedná se o nic jiného, než o data z minulých projektů, které nám mohou poskytnout lepší náhled na situaci a lépe tak odrážet realitu. Výstupem této části je sada reportů z metrik, ale také si tímto postupně budujeme kvalitní databázi historických dat. Na samém konci procesu dostane projektový manažer díky těmto metrikám cenné informace nebo v lepším případě i návrhy pro zlepšení situace a může tak učinit důležitá rozhodnutí v plánování projektu upravením jeho cílů.



Obrázek 3: Zavedení metrik [1]

Pro shrnutí bych zde uvedl výčet opakujících se aktivit, které se vztahují k měření a hlavně zlepšení projektů a procesů. Patří zde například:

- Stanovení krátkodobých a dlouhodobých cílů
- Identifikace a analýza možných rizik
- Vytvoření plánů, jejich koordinace a provádění
- Vytvoření motivace pro kolektiv, aby plnily své úkoly
- Předpovídání a odhadování směru vývoje procesů a produktů vzhledem k cílům
- Průběžná kontrola a porovnání výsledků metrik s cíli, zda jde vše dle plánu

- Nastavení a analýza metrik
- Vyhodnocení výkonnosti a kvality projektů nebo procesů
- Zjištění významných odchylek
- Identifikace a realizace nápravných opatření
- Porozumění a souhlas se změnami
- Ocenění/potrestání za provedené výkony

Projekt často kontroluje více uživatelů a ti mají různé uživatelské role. Toto by se mělo promítnout i do metrik samotných, kde každý z uživatelů dostane k dispozici pouze metriky, které jsou pro něj důležité. Platí zde pořekadlo, že „někdy méně je více“, protože zahlcený uživatel větším počtem metrik může být často frustrován a může ztratit dobrý přehled o situaci.

Za zmínku také stojí, že metriky se mohou lišit periodou. Ta určuje, za jaký časový údaj získáme výsledky, které můžeme vizualizovat a vyhodnotit. Kratší interval může být například den, týden, zatímco delší mohou zobrazovat údaje až jednou za kvartál případně rok. S tímto souvisí i předpovídání, které je v tomto ohledu velmi důležité a není radno ho podceňovat, protože u metrik trvajících delší časový úsek často nemáme v průběhu žádné informace. K této aktivitě využíváme historických dat z námi vybudované báze znalostí a zkušeností. Výsledkem dobrého předpovídání je zmírnění rizik a získání plánů, které není potřeba v budoucnu zásadně měnit.

Je dobré začít pouze s minimem metrik, které budou pro naše projekty klíčové. Tyto metriky by měly dodržovat určitý standard a umožnit nám tak porovnání, automatizaci a znovu použitelnost skrze všechny projekty. V této základní sestavě by měly být metriky, které nám umožní například sledovat velikost projektu a jeho růst. Stejně tak musíme mít přehled nad průběhem a uplynulým/zbývajícím časem. Protože je naším záměrem dodat produkt nebo službu v dobré kvalitě, tak i tady musíme mít nasazené metriky. Z pohledu managementu je rovněž důležité sledovat vynaloženou snahu a rozpočet.

4 Studované metody a metriky

Základem dobrého plánování je správné využití dostupných zdrojů. Následující kapitola se věnuje hlavně personálním, lidským zdrojům. Zmíním zde základy pro metody a metriky, které měří vynaložené úsilí a velikost softwarového díla, jež jsem měl na starosti.

4.1 Odhad vynaloženého úsilí

Nedílnou součástí softwarového managementu a projektového řízení je odhad vynaloženého úsilí, které umožnuje softwarovému manažerovi sledovat personální zdroje. Poskytuje přehled o zaměstnancích, kteří se nemalou mírou podepisují hlavně na vývoji, ale také nákladech projektu. Vynaložené úsilí je definováno jako celková suma času pro jednotlivé úkoly, který se podílí na vytvoření funkčního produktu nebo služby. Nejčastěji se vyjadřuje v jednotkách času – ve dnech, či hodinách. Protože je toto úsilí spojeno s lidskými zdroji, tak není neobvyklé ani použití jednotek, jako jsou člověkodny, člověkohodiny apod.

Ukazatelé vynaloženého úsilí mohou být použity ve všech úrovních projektového řízení pro měření aktuálního stavu oproti očekávanému plánu. Velice důležitou funkcí vykonávanou softwarovým managementem je stanovení počtu zaměstnanců pro konkrétní úlohu. Sečtením všech úloh tak získáme celkový počet potřebných zaměstnanců pro dané období na konkrétním projektu. Díky tomu jsme schopni lépe řídit naše personální zdroje, lépe koordinovat práci a zvýšit tím celkovou efektivitu organizace. Myslím, že netřeba zdůrazňovat, že metriky tohoto typu mají hodně společného s metrikami nákladů. Podle konvencí se ovšem dodržuje pravidlo, že metriky vynaloženého úsilí zastřešují hlavně problematiku lidských zdrojů.

Popis	Vynaložené úsilí [hodiny]
Konzultace funkčních požadavků se zadavatelem	50
Zavedení metriky pro sledování počtu chyb (5 osob)	30
Otestování modulu pro sběr externích dat	70
Stanovení vah pro metriky funkčních bodů	20
Vynaloženého úsilí celkem:	170

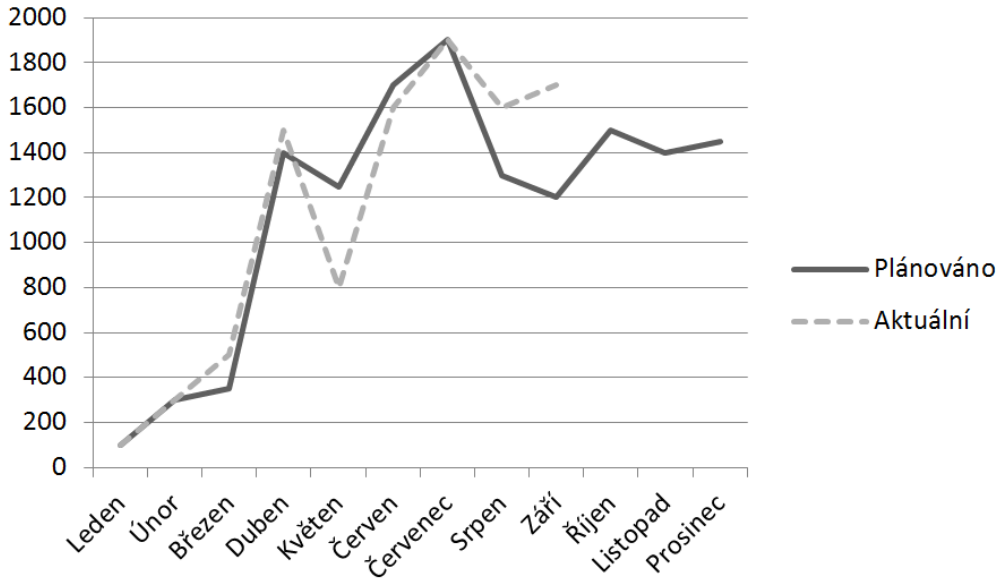
Tabulka 1: Příklad popisu a ohodnocení vynaloženého úsilí

Ačkoliv se může zdát, že evidované položky v tabulce jsou banální, tak jejich názornost a vypovídající hodnota je vysoká. Tabulka zobrazuje úkoly a jejich plánovaný čas označován také jako vynaložené úsilí. Odhadnutí těchto hodnot není ale jednoduché a vyžaduje zkušenosti a expertní znalosti.

Metrika vynaloženého úsilí se skládá z těchto elementárních kroků:

1. Softwarový manažer stanoví pro jednotlivé úkoly, kolik budou stát úsilí (časová jednotka). Vznikne tím plánované úsilí. Využívá se zde hojně historických dat z podobných projektů.

2. Během celého životního cyklu sbíráme data, která se týkají výše zmíněných položek. Dále je budu označovat jako aktuální úsilí. Můžeme také přidávat další plánované úsilí.
3. Za námi určenou jednotku vždy porovnáme plánované úsilí s aktuálním a zjistíme tak, jak na tom jsme s využitím personálních zdrojů a můžeme tak stanovit náklady.



Obrázek 4: Příklad vizualizace vynaloženého úsilí

Pokud provedeme výše zmíněný proces plánování a sběru dat, tak nic nebrání metriku vizualizovat. Jednu z možných interpretací reprezentuje grafické vyobrazení metriky pomocí grafu (obrázek 4). Díky tomu, že na ose X jsou naneseny měsíce a osa Y zobrazuje počet hodin, nám nic nebrání nanést křivky plánovaného a aktuálního vynaloženého úsilí. Vidíme zde například, že v květnu je aktuální úsilí daleko menší, než plánované, a proto v tomto měsíci jde vše nad očekávání. V září to už není tak dobré a z toho usuzujeme, že plánování i tým pracující na nějakém z úkolů selhal. V této části by měl projektový manažer učinit nápravné akce a dostat projekt zpět na vytyčenou cestu.

4.2 Odhad velikosti

Bez odhadu velikosti softwaru bychom těžko stanovili odhad vynaloženého úsilí, které jsem zmiňoval v předešlé části. Jedná se totiž o kritický vstup pro fázi plánování a řízení projektu. Softwarový manažer tímto druhem metrik sleduje plánovanou a aktuální velikost softwaru. Může sledovat celou řadu aspektů. Může se například jednat o trendy ve velikosti kódu samotného, odchylky mezi aktuální a plánovanou velikostí softwaru nebo odchylky mezi jednotlivými verzemi. Na základě změn velikosti softwaru během času můžeme odvodit jeho stabilitu velikosti. Ta nám poskytuje informace o tom, zda jsou požadavky pochopeny a zda jsou kompletní, jestli byl vytvořen pečlivě návrh a také odhaluje samotné schopnosti softwarových vývojářů plnit požadavky při dodržení stanoveného rozpočtu. Metriky pro odhad velikosti se dají používat během celého životního cyklu softwaru. Sběr dat je prováděn a vyhodnocován v intervalu minimálně jednou za měsíc. Nestabilita velikosti softwaru v jiné, než rané fázi vývoje signalizuje, že je potřeba učinit nápravná

opatření. Dále nastíním pár možných přístupů k určení velikosti softwaru a tím pádem i k získání přehledu o vynaloženém úsilí.

4.2.1 Lines of Code

Jedním z nejjednodušších způsobů měření velikosti softwaru je počítání řádku zdrojového kódu. *Lines of Code metriky* by se daly rozdělit na dvě základní skupiny. První z nich měří fyzické řádky (označována LOC) včetně komentářů i prázdných řádků za předpokladu, že jejich počet nepřekročí 25 % z celkového kódu. Druhá skupina (LLOC) počítá pouze logické konstrukce příkazů a bere je jako celek. Je tak složitější na nasazení. I když jsou tyto metody často využívány k odhadům velikosti softwaru, tak ne vždy musí mít vypovídající hodnotu. Je totiž běžné, že mezi softwarovými vývojáři jsou propastné rozdíly a ne vždy tak platí, že čím více řádků zdrojového kódu, tím bude lepší kvalita nebo větší funkcionalita softwaru. Další problém tvoří moderní nástroje, které automaticky vygenerují velké množství kódu během pár kliknutí myši. Také jeho opětovné použití zkreslí realitu.

4.2.2 Functional Size Measurement

Další možná varianta spočívá v měření velikosti funkčních požadavků kladených uživateli na vyvíjený software. Měření funkční velikosti softwaru je tak v kontrastu s měřením počtu řádku zdrojového kódu. Funkční požadavky totiž nejsou závislé na tom, jak je přesně software vytvářen. Jedná se o logický pohled na vyvíjený software, který neřeší fyzickou či technickou stránku věci.

Nejběžnější používanou metodou jsou *funkční body* (*Function Points*), označované jako *FP*, které přinášejí vylepšené odhady, lepší pochopení projektu a jeho údržbu. Protože využívají funkčních požadavků, tak mohou často pomoci při řízení změn požadavků kladených na projekt.

Stejně jako každý programátor má jinou výkonnost, tak se mezi sebou liší funkční požadavky. Některé jsou méně náročné, jiné zase vyžadují vynaložit větší úsilí k jejich splnění. Základní myšlenka funkčních bodů spočívá v ohodnocení a nastavení vah jednotlivým funkčním požadavkům. Jejich sečtením získáme hodnotu, která nám značí velikost a reprezentuje potřebné množství vynaloženého úsilí.

Aplikace metriky funkčních bodů se může lišit metodu od metody, ale základní princip je pro všechny shodný. Z každého funkčního požadavku získáme komponenty, které závisí na typu uživatelské akce – transakční a datové. Metrika se skládá z identifikace, klasifikace a nastavení vah těmto jednotlivým komponentám.

Do transakčních komponent spadají uživatelské vstupy, výstupy a dotazy. Datové komponenty řeší pro změnu interní logické soubory a rozhraní k externím souborům. Počet položek pro jednotlivé typy komponent může být libovolný pro každý funkční požadavek. Přidáním vah jednotlivým položkám komponent zajistíme přidání další užitné informace. Budeme tak moci rozlišit jednotlivé požadavky, co se týče jejich velikosti a tím pádem i vynaloženým úsilím. Nejlepší bude si toto ukázat na názorném příkladu (tabulka 2).

Jedná se o použití metriky funkčních bodů na konkrétním funkčním požadavku. První sloupec tabulky obsahuje celou pětici komponent, za kterou následuje jejich souhrnný počet. Dále zde najdeme nastavení vah, které může být pro jednotlivé komponenty rozdílné. Nicméně pro všechny funkční požadavky musí být tyto váhy už stejné, aby nedocházelo k nekonzistenci a hlavně zkreslení výsledku. Poslední sloupec obsahuje počet funkčních bodů, který se získá vynásobením počtu

jednotlivých komponent s jejich odhadovanou váhou složitosti (vyznačena v tabulce zeleně). Sečtením všech řádků tabulky (FP ode všech komponent) získáme celkovou sumu funkčních bodů, pro jeden funkční požadavek kladený uživatelem. Pokud bychom tento postup aplikovali na všechny tyto požadavky, tak tím získáme celkovou sumu funkčních bodů pro celý softwarový projekt. Jedná se sice „pouze o číslo“, ale má daleko větší vypovídající hodnotu, než například LOC metriky.

		Váha			
Měřená komponenta	počet	jednoduchá	normální	složitá	počet FP
Uživatelské vstupy	4	4	5	7	20
Uživatelské výstupy	7	5	6	9	63
Uživatelské dotazy	5	3	5	8	15
Interní logické soubory	14	8	12	17	238
Rozhraní k externím souborům	3	6	8	13	24
Funkčních bodů celkem:					360

Tabulka 2: Příklad použití funkčních bodů

4.2.3 Příklad vyhodnocení sady metrik

V následující části budu demonstrovat vyhodnocení sady metrik. V tabulce vidíme sedm různých metrik s tím, že některé spolu víceméně souvisejí (tabulka 3). Začal bych u metriky logických řádků kódu, která plánovala jejich počet na 100 000. Aktuální hodnota je daleko vyšší a ještě k tomu není zdrojový kód zcela kompletní (např. chybí naimplementovat některé moduly). Poněkud reálnější náhled na situaci nabízí funkční body, které víceméně korespondují s procentem kompletace zdrojového kódu.

Metrika	Plánovaná hodnota	Aktuální hodnota
Velikost [LLOC]	100 000	120 000
Velikost [FP]	350 000	315 000
Kompletní zdrojový kód [%]	100%	85%
Kompletní testy [%]	85%	25%
Počet defektů v návrhu	5	2
Počet defektů v modulech	15	1
Vynaložené úsilí [člověkodny]	1 800	2 100

Tabulka 3: Příklad sady metrik

Počty defektů v návrhu jsou sice menší než plánované, ale je třeba také sledovat pokrytí kódu testy a jejich procento. Je jasné, že není možné otestovat zcela celý kód, ale stěžejním částem by se měl věnovat dostatečný prostor. V našem konkrétním případě je počet kompletních testů pouze 25 %, a tak počet defektů v modulech a návrhu může být v reálu daleko větší, než ukazuje metrika. Vynaložené úsilí a procento kompletního zdrojového kódu, případně funkčních bodů, rovněž ledacos napoví. V tabulce vidíme, že vynaložené úsilí přesahuje plánovanou hodnotu a při tom softwarové

dílo ještě není zhotoveno. Může to indikovat například malou výkonnost týmu, způsobenou špatnou motivací. Rovněž pokud si teprve budujeme databázi dat, tak je možné, že naše odhady nebudou zcela přesné.

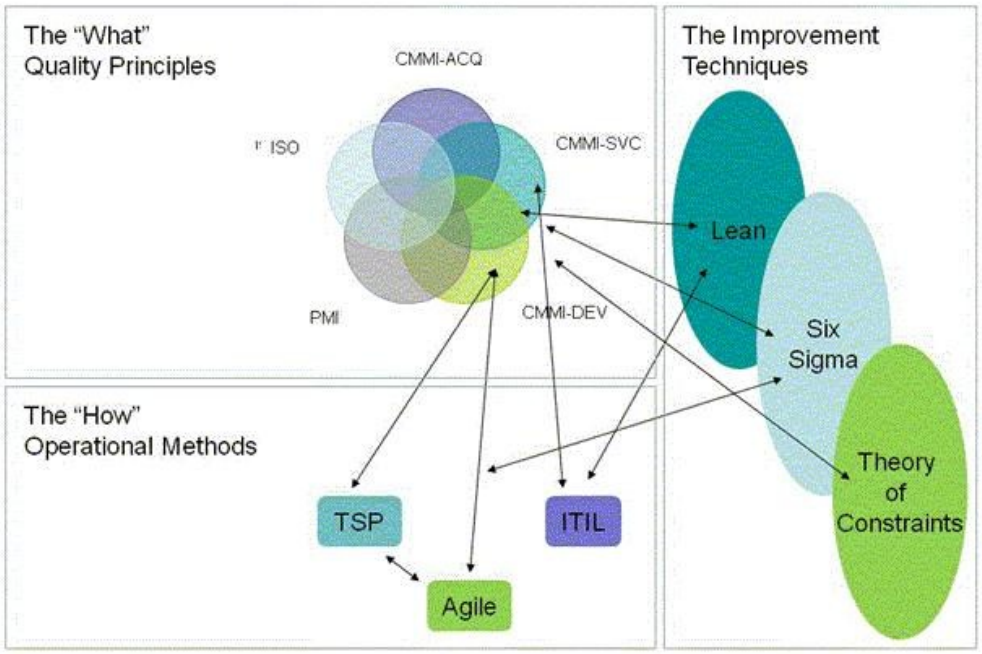
Tímto bych chtěl zdůraznit, že metriky jsou dobrým pomocníkem při sledování softwarového díla a vytváření nápravných opatření. Je nutné je nejenom dobře aplikovat a zavést, ale i správně vyhodnotit. Mnohdy je potřeba brát v potaz stav více metrik najednou. Stejně tak se mohou lišit průběhy jednotlivých metrik během životního cyklu softwarového díla, ale to bych zmínil více podrobně v kapitole 8.

5 Moderní metody vývoje softwaru

V předchozích kapitolách byla zmíněna motivace a základní teorie metrik. Metody vývoje softwaru s touto problematikou úzce souvisí. Vzhledem k souvislostem, které se týkají *aplikačního frameworku* a analyzovaných metrik, je nemohu vynechat. V dnešní době existuje celá řada metod, umožňujících nám vyvíjet kvalitní software s dodržením rozpočtu a termínů. Než se rozepíši o nejznámějších metodách současnosti podrobněji, tak stručně uvedu, jak se dají obecně rozdělit. Toto dělení pak krátce popíši a přidám mu také historické pozadí.

5.1 Dostupné metody

Za vznik počítačů může člověk, který si chtěl ulehčit nebo zjednodušit některé úkony, které by mu zabraly velké množství času. Na některé z nich by ani nemusel sám stačit. Tyto účely plní software, bez něhož by počítače postrádaly smysl. V samotných počátcích vznikal triviální software s malou velikostí a časovou náročností vývoje. Tento software často vyvíjela pouze jedna osoba a díky jeho relativní jednoduchosti ho nebyl problém vyvíjet ve stylu pokus/omyl. Na poskytované funkce softwaru uživatelům časem stále více narůstaly nároky a o implementaci komplexních požadavků se dnes starají velké týmy vývojářů. To, že je třeba tyto týmy koordinovat a měřit jejich výsledky, jsem zmiňoval v minulých kapitolách. Nevyjádřil jsem se ale o základních metodách, jakými se dá vést a řídit vývoj softwarového díla.



Obrázek 5: Dostupné metody [2]

V dnešní době existuje celá řada zásad, doporučení, metod a standardů, podle kterých můžeme vyvíjet softwarové dílo (obrázek 5). V raných 80. letech minulého století Letecké Síly Spojených Státu Amerických financovaly studii, jejímž úkolem bylo zjistit, proč mnoho softwarových kontraktů

časem překročí svůj rozpočet. Závěr byl jednoznačný: špatné postupy nebo procesy při vývoji. Postupem času tak vznikly techniky vedoucí ke zlepšení vyvíjeného softwaru, principy kvality a v neposlední řadě provozní metody. Většina organizací pro lepší úsporu prostředků používá jen určitou podmnožinu z těchto metod. Nicméně i přes zvýšenou počáteční režii se to v pozitivním slova smyslu může projevit ve výsledku. Společnosti tedy ve snaze zdokonalit své procesy a projekty aplikují výše zmíněné techniky a metody. Spadají zde například i metriky, které sbírají, analyzují a vyhodnocují data, podle kterých můžeme učinit nápravná opatření.

Dále se budu v diplomové práci věnovat výhradně agilním metodám. Jeden z důvodů je i ten, že větší část námi studovaných metrik a hlavně nástrojů na nich byla založená.

5.2 Agilní metody

Agilní vývoj softwaru je tvořen skupinou metod, založených na iterativním a inkrementálním způsobu vývoje. Za nejvyšší prioritu si klade uspokojit potřeby zákazníka dodávkami kvalitního softwaru. Díky iterativnímu a inkrementálnímu typu vývoje není problém doručovat software v krátkých časových intervalech. Hlavním měřítkem pokroku je brán fungující software. Běžně nejsou změny požadavků během vývoje a hlavně na jeho konci ostatními metody vítány. To ovšem není případ agilních metod, které tak nabízejí výhody pro zákazníka. Správným užitím zmíněných metod se vyvarujeme nepříjemným překvapením.

Metody se zakládají na úzké spolupráci mezi zákazníkem a vývojáři, která probíhá pokud možno denně v průběhu celého projektu. Důležitá je i konzistentnost informací, které má tým vývojářů k dispozici. Není žádným tajemstvím, že nejlepší formou komunikace je osobní kontakt. Během celého životního cyklu by měli uživatelé i vývojáři udržovat tempo, které se dá díky vývoji v krátkých časových úsecích dobře měřit a upravovat. Tyto pravidelné intervaly umožní týmům vývojářů určit svou efektivitu, případně vyladit a přizpůsobit své chování. Je rovněž dobré dopřát týmu jistou míru volnosti, protože právě při ní vznikají ty nejlepší nápady. Agilní vývoj softwarového díla se nehodí do všech typů organizací. Tato metoda se používá spíše u menších týmů ve většině projektů.

5.3 Historie agilních metod

Kořeny agilních metod můžeme najít v 30. letech minulého století, kdy neexistovaly počítače a internet se službami s ním spjatými. Přesto i v této době se ovšem používaly základy těchto metod. Logicky první použití nemohlo být v softwarové oblasti, nýbrž ve strojním průmyslu. Zhruba v polovině minulého století se začaly používat základy metod iterativního a inkrementálního vývoje (IID) ve velkých organizacích jako DOD (Department of Defence), NASA (National Aeronautics and Space Administration) a US Air Force. Pořád se ale nejednalo o použití na softwarovém projektu.

Poprvé byly tyto metody použity k vývoji softwaru v padesátých letech minulého století. Mnoho z těchto „raných“ projektů bylo experimentálních a explorativních a sdílely mnoho atributů s dnešními agilními metodami. Procesní povaha armádních standardů, které měly pevně stanovenou cenu zakázek i pro komplexní systémy, vyžadovala zpracování velkých a složitých dat a dodání v určité časové lhůtě.

V roce 1976 vyšla kniha *Software Metrics* (Tom Gilb), která se věnovala agilnímu a adaptivnímu vývoji v iteracích. Postupem času softwarové inženýrství vyzrálo a vzniklo více formálních aplikací metod IID. V roce 1985 poprvé zmiňuje Barry Boehm spirálový model ve vědeckém článku *The Spiral Model of Software Developement*.

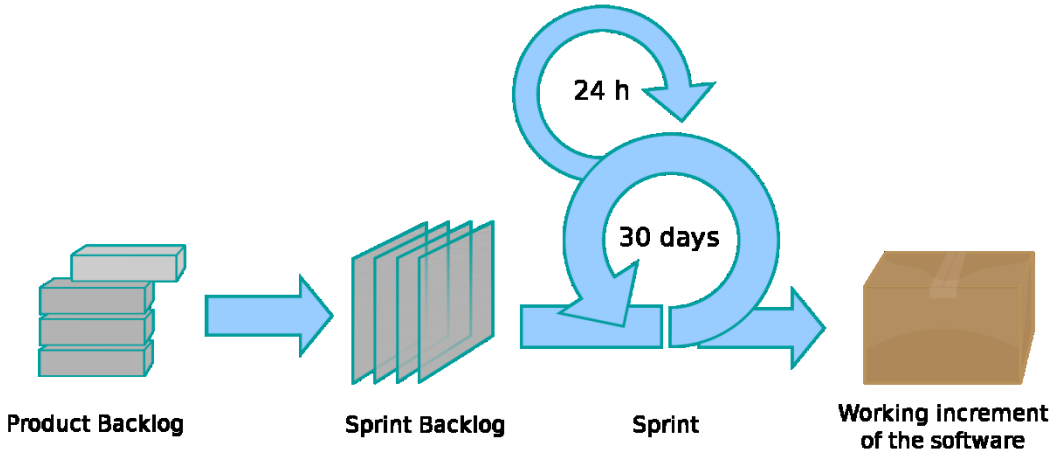
Během 90. let minulého století získal IID širší přijetí softwarové komunity a vznikaly další formy jako *rapid prototyping*, *rapid aplication developement* (RAD) a *rational unified proces* (RUP). V roce 1996 začali obhájci IID – Ron Jeffies a Kent Beck používat techniku zvanou *extreme programming* (XP) ve společnosti Chrysler, zatímco v Singapurské bance United Overseas spustily funkčně řízený vývoj (FDD). XP se stal postupem času velmi oblíbeným a patřil k nejznámějším metodám z agilní rodiny.

Už před koncem minulého století bylo jasné, že komunikace tváří v tvář, interakce se zákazníky, práce v menších týmech a včasné a časté dodávky softwaru, bude hlavním klíčem k úspěchu. V tomto období také vznikla metoda, která je v posledních letech na velkém vzestupu – *Scrum*.

S větším rozšířením agilních metod přišla potřeba mít nad nimi přehled a umět je koordinovat. To vyústilo v konferenci, na které se sešli všichni významní autoři agilních metod, kteří byli zodpovědní za jejich teorii a aplikaci. Výsledkem tohoto setkání bylo vydání *Manifesto for Agile Software Developement*. Jedná se o soubor základních myšlenek a filosofie, podle kterého bychom se měli řídit při vývoji softwaru. V dnešní době spravuje agilní metody a jejich standardy nezisková organizace *Agile Alliance*.

5.4 Scrum

Scrum je relativně nová metoda agilního programování z přelomu století, které se v posledních letech dostává čím dál větší pozornosti. Umožňuje menším týmům vývojářů v krátkých iteracích vytvářet kvalitní produkty s dobrým odhadem ceny a dodržením požadavků zákazníka. Oproti například vodopádovému modelu, umožňuje zobrazit výsledky práce mnohem dříve (byť se může jednat pouze o část plánované funkcionality). Ačkoliv název *Scrum* není akronym, tak některé společnosti jej píší s velkými písmeny jako *SCRUM*.



Obrázek 6:Scrum [3]

Před vysvětlením procesu *Scrum* (obrázek 6) musím objasnit hlavní uživatelské role, které zde figurují:

1.) *ScrumMaster* dohlíží nad celkovým procesem. Odstraňuje překážky a chrání tým vývojářů před vnějšími vlivy. Pomáhá týmu dosáhnout k jeho cílům a motivuje ho k lepším výsledkům.
2.) *ProductOwner* zastupuje vlastníka produktu. Má na starosti definování vize projektu a priorit úkolů.
3.) *Tým vývojářů* si sám rozděluje práci mezi jednotlivými členy. Jedná se například o analýzu, implementaci, testování atd.

Prvním artefaktem Scrumu je *Product Backlog*, do něhož se postupně umisťují všechny *user stories* bez hierarchického členění. Jedná se o požadavky kladené na software samotnými uživateli. *ProductOwner* určí, které položky s projektem souvisí a které nikoliv.

Software se vyvíjí v krátkých iteracích, které se zde nazývají sprinty. Ty trvají většinou 2 až 30 dnů. Před započetím práce na sprintech je každé ráno porada – *Scrum daily meeting*. Trvá pouze okolo čtvrt hodiny a *ScrumMaster* položí každému členovi týmu trojici základních otázek: Co jsi udělal od včerejška? Co máš v plánu dělat dnes? Brání ti v tom něco?

Sprint backlog je další artefakt, který obsahuje pouze část *user strories*, které budou na konci sprintu hotovy. Všechny položky jsou oceněny, co se týče časové náročnosti a je jim nastavena priorita. S ubývajícím časem sprintu by měl i ubývat odhadovaný čas práce na jednotlivých položkách. Na konci sprintu by měly být všechny *user strories* naimplementované a plně otestované.

V případě, že tomu tak není, můžeme učinit nápravné akce. Výsledkem každého sprintu by měl být vždy plně funkční software, který implementuje všechny požadavky pro daný sprint.

Jedním z hlavních důvodů velké obliby Scrumu je možnost vizualizace průběhu každého sprintu denně. Graf nazýváme *BurnDown Chart* (kapitola 7). Data k jeho vytvoření jsou dostupná, protože jednotlivé *user strories* jsou časově ohodnoceny a na konci každého dne tým vývojářů zaznamená změny jejich časového plnění. Jednoduše tak můžeme porovnat plánovaný čas s tím aktuálním. Hned při prvním pohledu vidíme, zda jde vše podle plánu.

5.5 Možné dělení metrik u Scrumu

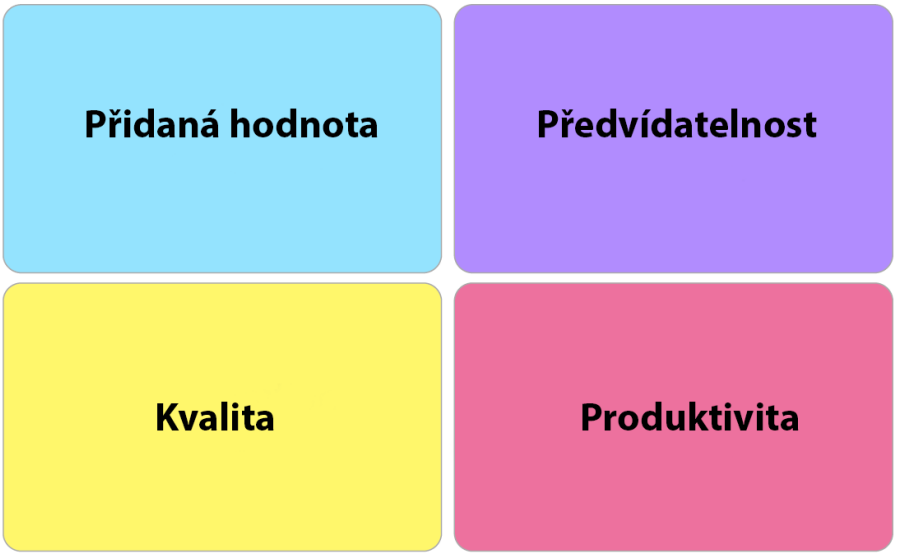
V následující části bych se opět vrátil k metrikám, které mohou být využívané u agilních metod, konkrétně u *Scrumu*. Chtěl bych zde hlavně zmínit možné rozdělení do jednotlivých dimenzí (obrázek 7). Ty jsou celkem čtyři a dělí se na *Přidanou hodnotu*, *Předvídatelnost*, *Kvalitu* a *Spolupráci*. Nedá se říci, zda existují dobré nebo špatné metriky. Stejně se nedá říci, že každá metrika spadá pouze do jedné z těchto dimenzí. Záleží hlavně na tom, jak dobře dané metriky pochopíme a následně aplikujeme.

Metriky by neměly nahrazovat naše přemýšlení, ale jsou pouze jeho vstupem pro rozhodování. Pokud nám to může pomoci, tak si můžeme vytvořit i vlastní dimenze, které budou reprezentovat mírně odlišné charakteristiky.

5.5.1 Přidaná hodnota

Business Value Delivered měří cenu každého sprintu, která je založená na jednotlivých položkách *product backlogu*. Vysoké číslo je dobrý indikátor a může být také dobrou motivací pro tým. Vzhledem k tomu, že toto číslo běžně během celého sprintu postupně klesá, tak se paradoxně může stát pro tým demotivujícím a mělo by se s tímto aspektem počítat.

Další dvojici metrik tvoří *Customer Satisfaction Survey* a *Employee Satisfaction Survey*. Díky zpětné vazbě umožňuje průzkumu spokojenosti určit plno hodnotících kritérií. Průzkum spokojenosti zákazníka se tak například zabývá kvalitou, předvídatelností, podporou, nebo dokonce spokojeností s novými funkcemi. Druhý průzkum řeší pohled zaměstnance, kde je zajištěna jak kvalitativní, tak kvantitativní zpětná vazba. Řeší se tak otázka kvality práce, pracovní zázemí, týmová práce, dodržování procesu, definice výrobku nebo spokojenost. Je zjištěno, že spokojení pracovníci zvyšují produktivitu týmu.



Obrázek 7: Možné rozdělení metrik do čtyř dimenzí

5.5.2 Předvídatelnost

Enhanced Burndown chart (Kapitola 7) je, jak už jsem zmínil, základním kamenem *Scrumu*. Zobrazuje průběh práce na úkolech během jednotlivých sprintů. Často je spojován i s metrikami nákladů. *Velocity* neboli rychlost/spád je v podstatě podmnožinou metriky výše. Jedná se o přímku, která nám udává, kolik práce jsme schopni vykonat za jeden den. Toto se dá opět porovnávat s plány.

Cost per Story Point udávají cenu jednotlivých uživatelských příběhů (user stories). Určuje se na základě počtu lidí v týmu, délce sprintu a časové náročnosti úkolů. Podobnou metrikou je *Hours per Story Point*, která pro změnu sleduje celkový čas pro jednotlivé body příběhu. Měla by sloužit spíše k vyhodnocení předvídatelnosti.

5.5.3 Kvalita

Spadají zde například *automatizované testy* – jednotkové a funkcionální, které se provádějí v každém sprintu. Vysoké číslo úspěšných testů je dobrý indikátor, ale mělo by se ještě kombinovat

s metrikou pokrytí kódu testy, jinak jejich vypovídající hodnota rapidně klesá. Dále můžeme měřit počet defektů, které se objeví až po dokončení sprintu nebo vydání. Je dobré také měřit počet defektů, které jsou klíčové, aby se nám je podařilo napravit.

5.5.4 Produktivita

In Sprint Cycle Time měří dobu, za jakou jsme schopni splnit uživatelské příběhy. Čím je tento čas kratší, tím je produktivita vyšší a naopak. *In Sprint Work in Progress* udává, jak mnoho je rozpracované práce. Paradoxně nižší číslo je dobrým indikátorem, protože vede k vyšší propustnosti. Pro grafické znázornění průběhu práce na uživatelských příbězích použijeme *Sprint Completion Bar*. Rozděluje příběhy do barevně odlišených kategorií – Hotovo, Netestováno, Nenakódováno a Ztráta času. Dle počtu položek se stanoví procenta, která se promítnou na délce každého sloupce (Hotové položky by měly převažovat).

6 Přehled dostupných nástrojů

Při průzkumu stávajícího stavu na trhu nástrojů, které lze použít pro sledování procesů a projektů v organizacích, jsme narazili na jejich veliké množství. Zkoumali jsme jak volně šiřitelné, tak komerční nástroje. Dle očekávání jsme zjistili, že přidaná hodnota těch komerčních byla značně vyšší.

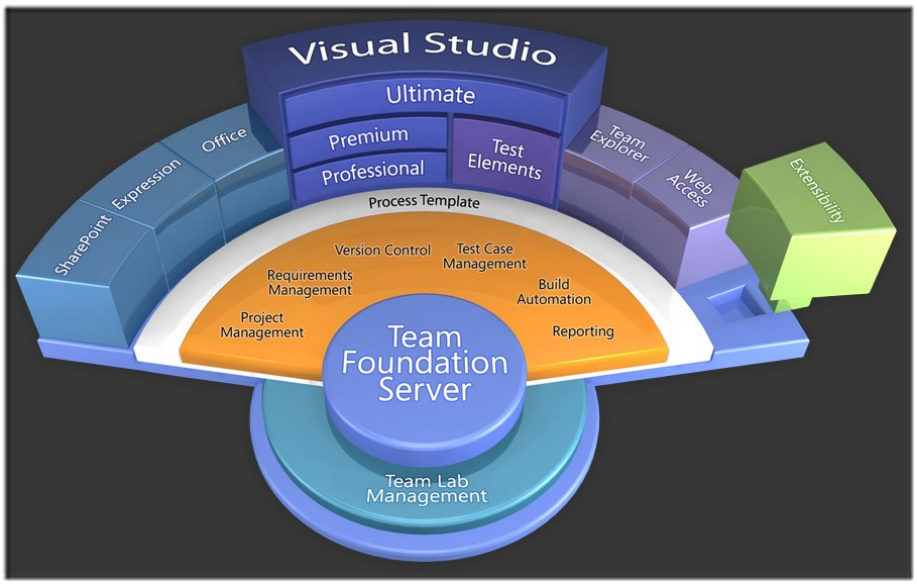
Většinu z komerčních nástrojů může uživatel bezplatně po nějakou dobu vyzkoušet. Toho jsme využili k průzkumu komplexnosti jednotlivých řešení a inspiraci při studiu dané problematiky. Než se pustím do rozboru řešení, které jsem měl na starosti, tak bych ještě zmínil kompletní seznam analyzovaných nástrojů.

Seznam studovaných nástrojů je následující:

- LogiReport [7]
- JasperForge [7]
- Mantis Bug Tracker [7]
- JIRA [7]
- Sprintometer
- Team Foundation Server

6.1 Microsoft® Visual Studio Team Foundation Server

Tento vyspělý nástroj z dílny softwarového gigantu Microsoft se zaměřuje na týmový vývoj softwaru a řízení životního cyklu při vývoji, který poskytuje mnoho základních funkcí jako správu verzí, kolekci dat, reporting nebo záznam projektu, kontrolu zdrojových kódů, záznamy pracovních položek a jiné.



Obrázek 8: Team Foundation server [4]

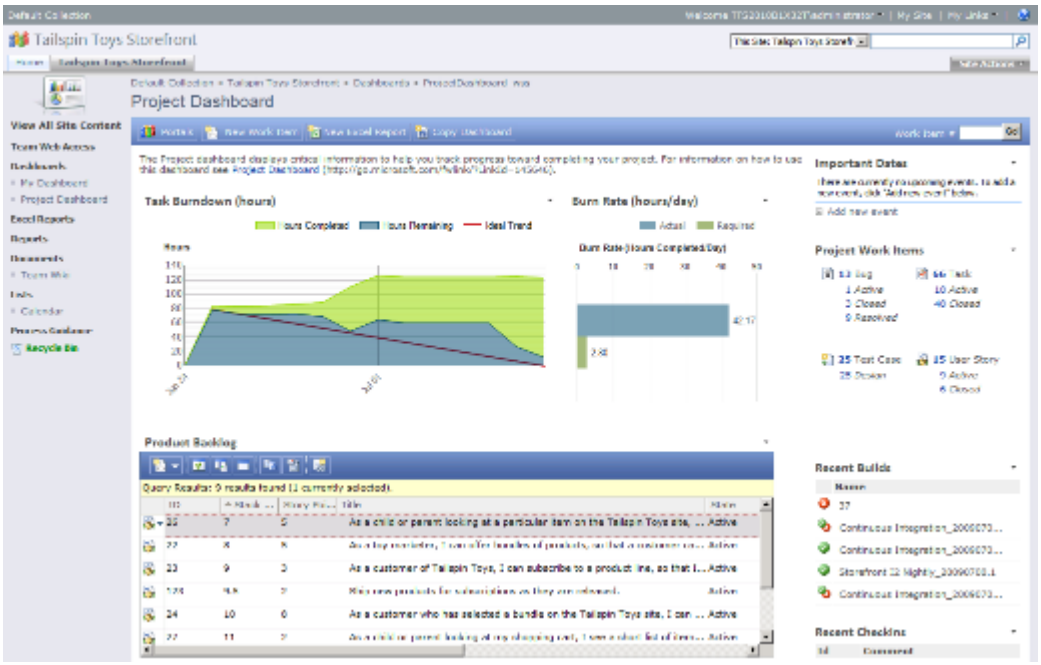
V podstatě se jedná o součást Microsoft Visual Studio, což je vývojové prostředí od Microsoftu. Jak je vidět na obrázku výše, tak *Team Foundation Server* (TFS) je opravdu robustní nástroj (obrázek 8), který zahrnuje velký okruh funkcí důležitých při vývoji softwarových děl. Mohlo by se tak zdát, že *TFS* je vhodný jen pro rozsáhlé organizace a nadnárodní společnosti, kde jsou týmy rozloženy různě po světě. Takovýmto společnostem *TFS* nabízí podporu architektury 64bitových serverů, či rozložení zátěže sítě. Na druhou stranu ani malé týmy o dvou či třech členech nepřijdou zkrátka. Takovéto malé týmy mohou využít zjednodušenou instalaci, díky které nebudou zatěžovány jak časově, tak velikostně náročnou plnou instalací *TFS*.

TFS můžeme spustit jak na klientském počítači s operačním systémem Windows Vista a novějším, tak na serverovém OS Windows Server 2003 a výše. Pro ukládání dat se pak používá Microsoft SQL Server. Vše je tedy uzavřeno na produktech společnosti Microsoft. Taková uzavřenost však nemusí být vždy na škodu, zvláště když se jedná o takto silný nástroj.

Microsoft do *TFS* zavedl podporu agilních procesů jako je *Scrum*. Tato metodologie vývoje umožňuje vytvářet software podle předem stanovených pravidel, kontrolovat jej a řídit tak, abychom se vyhnuli nežádoucím komplikacím či dokonce pozdnímu odevzdání hotového softwaru.

TFS nám poskytuje, stejně jako ostatní konkurenční nástroje, možnost vytvářet si vlastní nástěnky (*dashboards*). Jedná se o jakési kontrolní obrazovky, skládající se z různých panelů pro sledování stavu celého projektu či jeho dílčích částí. Dále nám zobrazují historické trendy vývoje, které nám mohou být užitečné. Z takto jednoduše vizuálně reprezentovaných metrik lze snadno detekovat potencionální problém a včas na něj zareagovat a upravit tak postup práce na projektu.

Díky integraci s produkty Microsoft Project a Microsoft Office Project Server získají obchodní partneři a správci projektu podrobný pohled na stav nedokončených projektů a pochopí, jak přispívají k plnění potřeb podniku. Taktéž lépe rozpoznají možnosti, jak zlepšit existující procesy. Dále *TFS* rozděluje jednotlivé uživatele na základě jejich rolí – vedoucí vývoje, designéři aplikací, vývojáři, testeři apod.



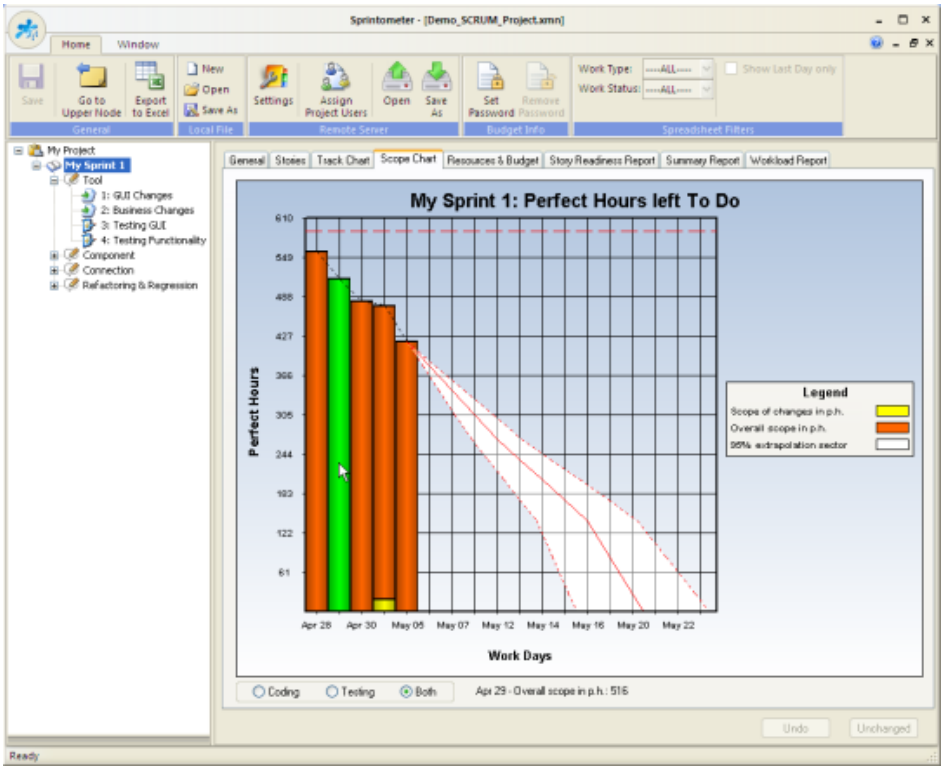
Obrázek 9: Team Foundation Server [4]

Další užitečné funkce a vlastnosti, které nám TFS předkládá:

- Paralelní vývoj aplikací
- Gated check-in - technologii pro ochranu před poškozením sestavení
- Analýza dopadu
- Řešení konfliktů
- Vlastní návrh workflow
- Správa testovacích případů
- Centrální úložiště všech součástí projektu
- Webový přístup k datům TFS
- Virtualizované testování

6.2 Sprintometer

Sprintometer je jednoduchá a uživatelsky přívětivá aplikace pro správu a sledování agilních projektů. Jedná se o freeware produkt, který lze použít pro správu *Scrum* a *Extreme programming* projektů. *Sprintometer* je nenáročná aplikace, která podporuje instalaci na různé varianty operačního systému Microsoft Windows. Od starších Windows XP (SP3) až po Windows 7. V oblasti serverových operačních systémů pak na Windows Server 2003 R2 (SP2) až Windows Server 2008. Podporuje dále dva druhy databází. Jsou to SQL Server 2005 (SP3) a SQL Server 2008 (SP1).



Obrázek 10: Sprintometer [5]

Předností, které v sobě tento nástroj ukrývá, je opravdu hodně. Mezi nejpodstatnější by se mohl zařadit vylepšený *Burn Down* a další užitečné grafy. Můžeme si tak vygenerovat velké množství reportů ze všech důležitých parametrů *Agile*. Velmi elegantně je zde řešeno přiřazení zdrojů k úkolům a uživatelským příběhům (*user stories*).

Při zavádění *Sprintometeru* nemá žádný vliv velikost naší organizace. Je velice variabilní ohledně velikosti a složení týmu. Vedoucí pracovníci tak mohou snadno sledovat jednotlivé sprinty či iterace vývoje a průběžně tak sledovat odchylku od stanoveného plánu. Dále lze snadno oddělit sledování vývoje od testování. Pokud se jedná o otázku uložení či načtení projektových dat, tato problematika je řešena velmi elegantně pomocí dat ve formátu XML. Naopak výsledná data reportů (grafy a tabulky) je možno exportovat do aplikace Microsoft Excel, která je pro tato data nejvhodnější.

Pokud jde o grafickou stránku, *Sprintometer* nám nabízí moderní, příjemné a snadno ovladatelné uživatelské prostředí ve stylu Microsoft Office 2007 (obrázek 10). Nechybí ani vysoká úroveň bezpečnosti. Veškerá data mohou být zaheslována a je k nim omezen přístup dle uživatelských rolí. Veškerá komunikace mezi klientem a serverem je zprostředkována skrze zabezpečené HTTPS spojení.

7 Metriky z Team Foundation Serveru

Následující část je věnována metrikám, které se vyskytují v *Team Foundation Serveru* od společnosti Microsoft. Tento nástroj byl pečlivě popsán v minulé kapitole (kapitola 6). *TFS* sloužilo, jako odrazový můstek pro vstup do světa metrik. Setkali jsme se zde poprvé s konkrétnější aplikací a vysvětlením metrik využívajících agilních metod. Práci jsme si s kolegy rozdělili na části, abychom co nejrychleji pronikli do problematiky.

7.1 Přehled agilních metod z TFS

Jak jsem uvedl výše, práci jsme si rozdělili s kolegy na tři části. První část měl na starosti Bc. Jakub Kamrla [7]. Na druhé části jsem pracoval já osobně. Třetí část prozkoumal Bc. Pavel Beneš.

7.1.1 První část

Během vývoje softwaru je běžné, že čas od času vznikají chyby. V první části přehledu agilních metod z *TFS* tak najdeme monitorování a sledování jejich trendů. Díky tomuto druhu reportů můžeme sledovat chyby, které tým našel a také vidět pokroky, které vedou k jejich nápravě.

Bug Status Report může být sestaven na základě aktivních (*active*), uzavřených (*closed*) a vyřešených (*resolved*) chyb v určitém časovém období. Z vytvořeného sloupcového grafu tak lze ihned vyčíst, zda se počet uzavřených a vyřešených chyb zvětšuje (naopak aktivních ubývá) a je vše na dobré cestě. Další variantou je koláčový graf, který nám pro změnu umožňuje sledovat chyby dle jejich priority nebo závažnosti. Stejně tak existuje ještě dvojice horizontálně orientovaných sloupcových grafů, kde jsou zobrazeny aktivní, případně vyřešené chyby pro jednotlivé členy týmu.

K účelům sledování rychlosti týmu při objevování a řešení chyb slouží *Bug Trends Report*. Na grafu zobrazuje průměrný počet chyb, které tým otevřel, řešil, případně uzavřel v zadaném časové období. Jestliže tým řeší chyby rychleji, než je objevují, tak počet aktivních chyb začne klesat. Ideální trend a hlavně méně chyb se může pozitivně projevit na stabilitě produktu.

Poslední metrika *Reactivation Report* určuje, jak efektivně tým opravuje chyby. Do grafu jsou vyneseny chyby, které byly uzavřeny a pak reaktivovány (znovuotevřeny). V ideálním případě by mělo uzavřených chyb přibývat, ale reaktivované položky by se měly držet co nejblíže nulové hodnotě a také by neměly úměrně růst s počtem uzavřených chyb.

7.1.2 Druhá část

V další části jsme se věnovali sestavením, jejich aktivitě, úspěšností a celkovými trendy. Tohoto se dá využít při sledování kvality a následné úspěšnosti sestavní pro daný tým v určitém časovém období.

Build Quality Indicators Report slouží k stanovení míry kvality softwaru pro jednotlivá sestavení, ve kterých je můžeme porovnat mezi sebou, co se týče kódu a úspěšných testů. V ideálním případě by většina testů měla být úspěšných a s minimálním počtem aktivních chyb, změn v kódu a neprůkazných testů.

Stav posledního sestavení zobrazuje *Build Success Over Time Report*. Jedná se o přehlednou vizualizaci sestavení pro jednotlivé dny. Každý stav sestavení má jinou barvu. Například bíla značí, že nebylo vytvořeno žádné sestavení. Dále zde najdeme barvy pro neúspěšné sestavení, úspěšné sestavení bez testů, neúspěšné testy a testy, které prošly a mají malé pokrytí. Zelená barva značí ideální situaci. To mimo jiné znamená úspěšné sestavení, kde jsou všechny testy úspěšné a stejně tak pokrytí kódů testy je dobré.

Build Summary Report zobrazuje souhrnně sestavení včetně informací ohledně testů, pokrytí kódu testy, změny v kódu a poznámky ohledně kvality. Obsahuje jak slovní popis, tak celkový průběh nebo kvalitu sestavení. Najdeme zde i procento úspěšných testů, a kódu, které pokrývají jednotlivé testy, nebo změny kódu udávané v počtu řádků.

Následující část sleduje zdraví projektu. Můžeme tak například zjistit množství vynaloženého úsilí týmu apod. *Burndown and Burn Rate report* slouží k určení míry rychlosti týmu, na základě odpracovaných a zbývajících hodin na projektu. Umožňuje porovnat aktuální trend oproti plánovanému a učinit tak potřebné rozhodnutí k případné nápravě.

K sledování průběhu úkolů je určen *Remaining Work Report*. Jako data opět poslouží odpracované hodiny nebo počet pracovních položek. Jedná se o pohled z jiného úhlu, než vyjadřuje Burdown graf.

7.1.3 Třetí část

V jednom z úseků třetí část jsme se věnovali pokroku iterací a uživatelským příběhům. *Status on All Iterations Report* umožňuje nahlédnout na pokrok stavů jednotlivých iterací. Dá se tak porovnat odhadovaný čas oproti realitě a také vidět počet a stav chyb. Vše je prezentováno jak vizuálně, tak numericky.

Stories Overview Report uvádí všechny uživatelské příběhy filtrované podle oblastí a iterace s přihlédnutím na priority. Je tak zachycena práce pro filtrovanou sadu uživatelských příběhů k aktuálnímu datu a také výsledky testů a počet chyb s nimi souvisejících.

Stories Progress Report zobrazuje celkový průběh jednotlivých uživatelských příběhů a jejich zbývajícího času k dokončení.

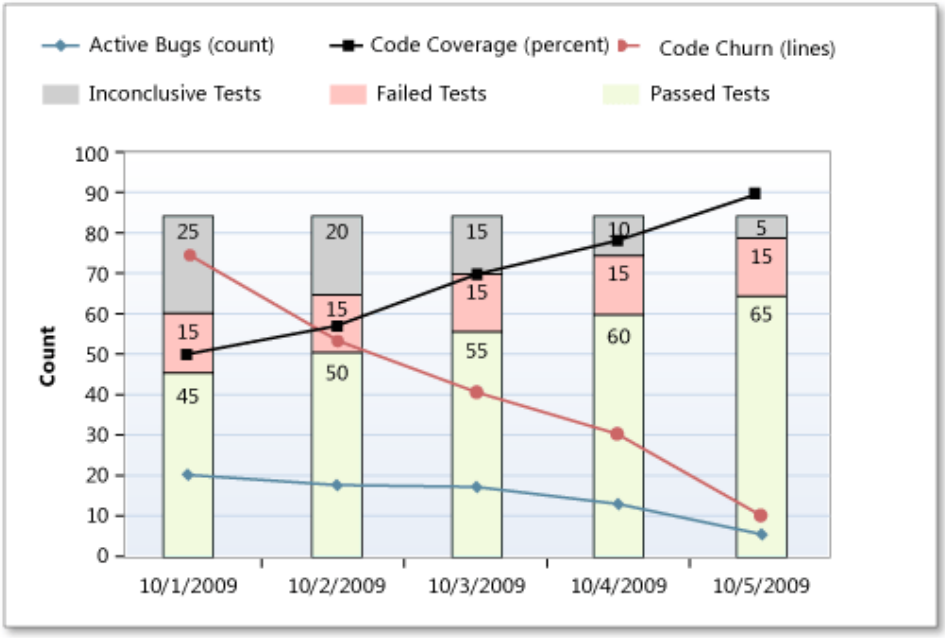
Samostatná část byla věnována určování přidané práce, kterou řeší metrika *Unplanned Work*. Na konci iterace tak můžeme zjistit množství přidané práce, která nebyla původně plánovaná.

Poslední část monitoruje aktivity spojené s testy – testovacími případy a jejich plány. Můžeme využít reportů k určení, jak si vede tým ve vývoji testovacích případů a také zjistit, jak tyto případy pokrývají uživatelské příběhy. *Test Case Readiness Report* slouží k zjištění, kolik testovacích případů tým definoval a kolik z nich je připraveno ke spuštění. Postupný nárůst definovaných případů by se tedy postupně měl „přelít“ do stavu *připraveno ke spuštění*. *Test Plan Progress Report* sleduje pokrok týmu v testování produktu. Kumulativně zobrazuje jednotlivé testovací stavy a jednoduše vidíme poměry mezi nimi. Na konci projektu by měla největší část obsahovat právě úspěšné testy.

7.2 Rozbor vybraných metrik

7.2.1 Build Quality Indicators Report

První z metrik *Build Quality Indicators Report* porovnává jednotlivá sestavení z více perspektiv (obrázek 11). Obsahuje informace o chybách a testech včetně pokrytí kódu testy a jeho změny. Díky těmto informacím můžeme zjistit kvalitu jednotlivých sestavení.



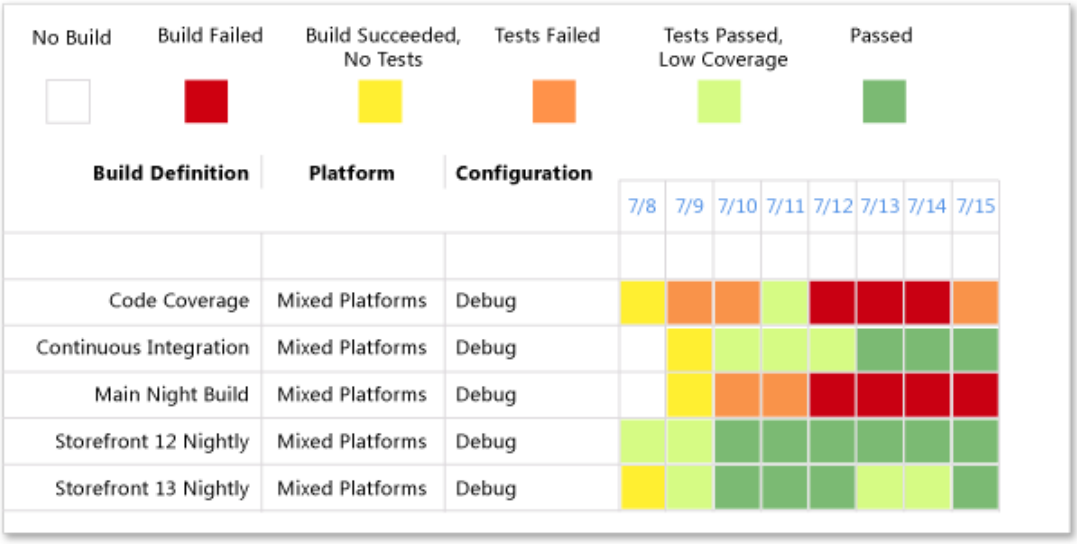
Obrázek 11: TFS – Build Quality Indicators Report [6]

Na ose X jsou vynášena jednotlivá sestavení. Osa Y zobrazuje jak aktivní chyby, změny v kódu (počet řádků) a pokrytí kódem (procenta), tak i testy – úspěšné, neúspěšné či neprůkazné (počet). V ideálním případě by procento úspěšných testů vůči ostatním mělo být co nejvyšší. Neúspěšné a neprůkazné testy by se měly před finálním vydáním blížit nule. Samo o sobě by ovšem toto nebylo dostatečné, protože je nutné také brát ohled na pokrytí kódu testy. Může nastat totiž případ, že všechny testy projdou, ale stále bude přítomno velké množství chyb. S tímto souvisí neprůkaznost testů a také jejich malá složitost. V ideálním případě postupem času ubývá chyb a stejně tak se zmenšují změny v kódu.

7.2.2 Build Success Over Time Report

K účelům sledování úspěšnosti sestavení slouží *Build Success Over Time Report*. Zakládá se na názornosti díky barevně odlišené vizualizaci stavů jednotlivých sestavení (obrázek 12). Řádky tabulky reprezentují sestavení. Sloupce, které představují dny, se označí barvou, která reprezentuje jeden ze šesti možných stavů sestavení. První dvojici těchto stavů tvoří případy, ve kterých nebylo vytvořeno nebo selhalo sestavení. U další trojice nebyl problém sestavení vytvořit. Liší se mezi sebou výstupem testů – žádné, neúspěšné nebo úspěšné, ale s malým pokrytím. Zelenou barvou jsou vyznačena úspěšná sestavení. Jedná se o sestavení, která mají v pořádku testy a pokrytí kódu testy je

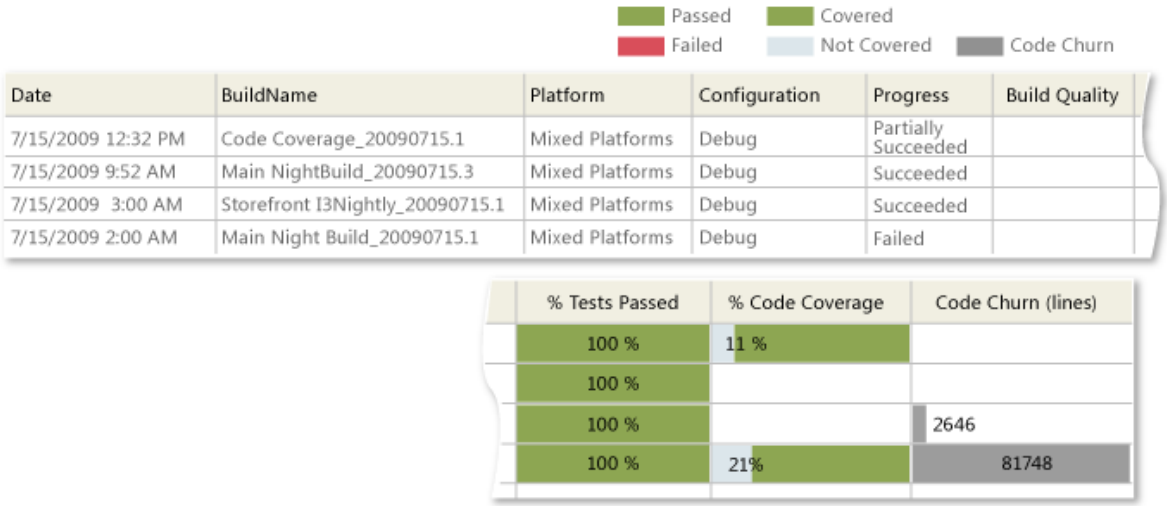
na dobré úrovni. Už při prvním pohledu tak lze díky barevnému odlišení jednoznačně identifikovat problémy sestavení a také trendy jejich vývoje. V ideálním případě by se barva výsledné mozaiky složené ze stavů a sestavení měla nést v zeleném tónu.



Obrázek 12: TFS – Build Success Over Time Report [6]

7.2.3 Build Summary Report

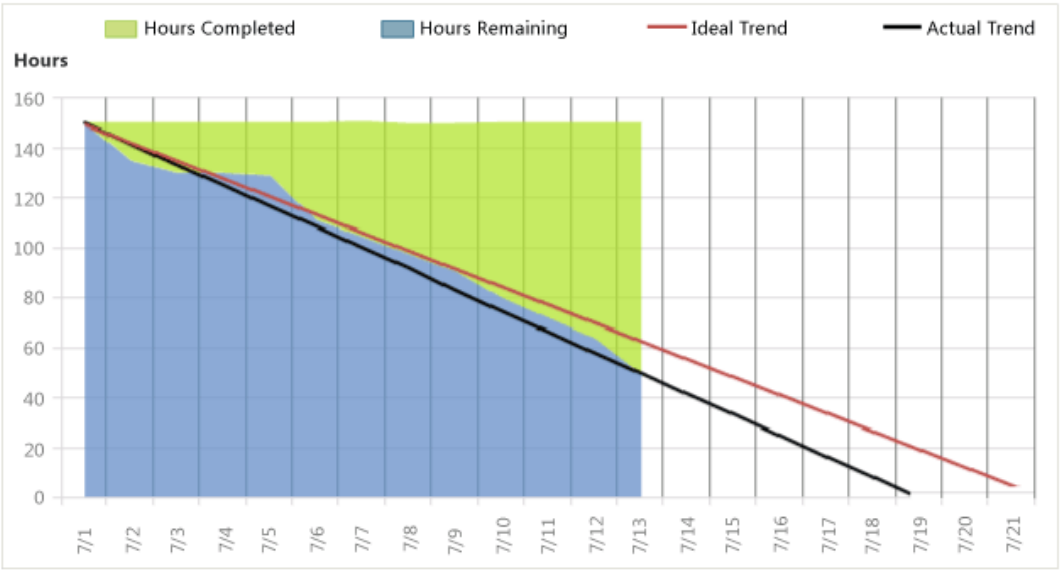
Build Summary Report kombinuje dvě předchozí metriky a nabízí tak opět mírně odlišný pohled. Sleduje opět sestavení a jejich kvalitu. V tabulce najdeme například informace o úspěšných a neúspěšných testech, nebo pokrytí a přírůstky kódu (obrázek 13). Je jasné, že pokud nabývá kód na objemu, tak bude jeho pokrytí testy klesat, pokud je nebudeme dodatečně vytvářet. O tom jestli je dané sestavení úspěšné, vyjadřuje atribut průběh (*progress*). Pokud nějaké sestavení selže, tak to zjistíme a můžeme zaujmout opatření.



Obrázek 13: TFS – Build Summary Report [6]

7.2.4 Burndown and Burn Rate Report

Následující metriky (obrázek 14) jsou stěžejní pro *Scrum*, jak bylo okrajově zmíněno již v 5. kapitole. Používají se hlavně při předpovídání vynaloženého času nad celkovým počtem uživatelských příběhů spadající do jednoho sprintu.



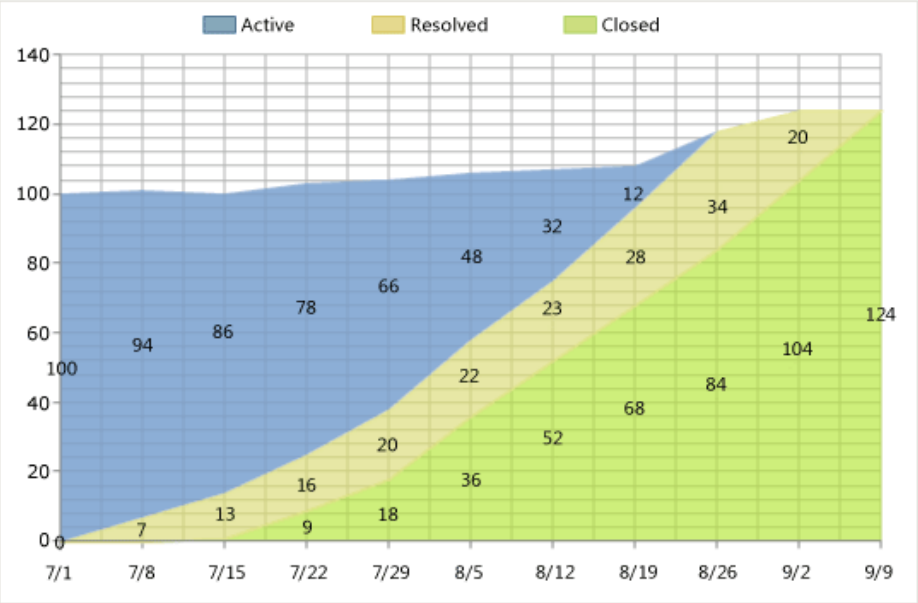
Obrázek 14: TFS – Burndown and Burn rate report [6]

Osa X reprezentuje jednotlivé dny sny sprintu. Na osu Y se vynáší celkový počet hodin. Protože jsou uživatelské příběhy předem oceněné, co se týče času, tak během jejich plnění víme přesně, kolik hodin ještě zbývá apod. Není tak problém v *Burndown reportu* zobrazit počet zbývajících a kompletních hodin v každém dni. Počet zbývajících hodin by měl být před koncem sprintu nulový.

Díky předem stanovenému odhadu času uživatelských příběhů a známé velikosti/výkonnosti týmu jsme schopni predikovat čas dokončení sprintu, případně ho můžeme stanovit dle naší potřeby. *Burn Rate* nám udává průměrně, kolik hodin práce jsme schopni vykonat za jeden den. Umožňuje porovnat ideální a aktuální trend. Aktuální trend je vždy spojnicí zbývajících hodin v počátku sprintu se dnem, kdy máme poslední data. Pokud je *Burn Rate* aktuálního trendu větší než plánovaného, tak vše stihneme v předstihu.

7.2.5 Remaining Work Report

Posledním zkoumaným reportem je *Remaining Work Report* (obrázek 15), který se podobá předchozí metrice. Může zkoumat odpracované hodiny a jeho interpretace je velmi podobná *Burndown reportu*. Další možností je sledování pracovních položek, které mají seskupené stavy v jednotlivých dnech. Postupně by se všechny pracovní položky měly přelít z aktivního do vyřešeného stavu. Prudký nárůst pracovních položek může sloužit jako varování, že se někde nachází problém.



Obrázek 15: TFS – Remaining Work Report [6]

8 Analýza webové aplikace

Následující kapitola je věnována analýze a návrhu *aplikačního frameworku* webové aplikace. Prvotní návrh počítal s lokální aplikací. Vzhledem k tomu, že by aplikace vyžadovala přístup na internet pro svá data, se ukázala tato myšlenka jako ne zcela ideální. Více o této lokální aplikaci píše *Bc. Jakub Kamrla* ve své diplomové práci [7]. Daleko lépe se tak jeví webová aplikace, která by byla dostupná uživatelům všude a bez nutnosti instalace. Nutnou podmínkou pro užívání by tak bylo pouze internetové připojení a webový prohlížeč. Díky této skutečnosti je možno přistupovat do webové aplikace nejen z PC, ale i z tabletu nebo mobilu.

8.1 Funkční požadavky

Úkolem bylo vytvořit *aplikační framework* pro metriky a měření kvality softwarového vývoje. Tento framework by měl agregovat data z více systémů a umožnit tak jejich jednotné vyhodnocení s důrazem na přehlednost. Projektový manažer tak nebude muset kontrolovat více systémů najednou, ale bude mít vše dostupné na jednom místě.

Ve webové aplikaci budou evidována data z více systémů pro řízení projektů. Tato data obsahují jednotlivé informace o vývoji softwarového díla. Jedná se o exportovaná data z databází těchto systémů ve formě *CSV* souborů.

Očekávané výstupy budou reprezentovat jednotně vizualizovaná data z jednotlivých systémů v námi navržené sadě metrik. Bude se jednat převážně o jejich interpretaci ve formě grafů a tabulek, které poskytnou koncovému uživateli lepší nadhled nad situací.

8.1.1 Okolí systému

Okolí webové aplikace tvoří uživatel a sledované systémy (obrázek 16). Uživatel interaguje s webovou aplikací ve dvou směrech. Může z ní jak data číst, tak i zapisovat. Webová aplikace dále pracuje s ostatními systémy v jednom směru, čerpá z nich data a agreguje je na jednom místě.

Obrázek 16: Kontextový diagram

8.2 Datová Analýza

Museli jsme zohlednit, že ve webové aplikaci budou agregována data z více systémů a připravit tak co nejvhodnější strukturu, na kterou by se daly ostatní systémy namapovat. Nejprve jsme museli analyzovat systémy, které budou zahrnuty do implementace. Rozhodli jsme se pro duo – *Atlassian JIRA* a *JasperForge*. Prozkoumali jsme ovšem i některé další systémy, abychom se mohli lépe

rozhodnout při tvorbě struktury dat webové aplikace. V následující části se budu věnovat čistě systému *JIRA*.

8.2.1 JIRA – analýza dat

Systém *JIRA* byl představen již v kapitole 6, ale bylo by jistě vhodné popsat, jaké atributy eviduje. Následující popis se bude věnovat jednotlivým atributům a také vysvětlením jejich významu.

Struktura dat:

Project, Key, Summary, Issue Type, Status, Priority, Resolution, Assignee, Reporter, Created, Update, Resolved, Affects Version/s, Fix Version, Components, Due Date, Votes, Watchers, Images, Original Estimate, Remaining Estimate, Time Spent, Work Ratio, Sub-Task, Linked Issues, Enviroment, Description, Security level, Progress, Σ Progress, Σ Time Spent, Σ Remaining Estimate, Σ Original Estimate, Label, Rank, Epic/Theme, Bonfire Url, Global Rank, Flagged, Bonfire URL

První skupina atributů se váže na rozdělení položek do více směrů a kategorií. Atribut *Project* značí, ke kterému projektu se záznam vztahuje. Jedinečný identifikátor zde zastupuje *Key*. K přidání krátkého shrnutí problému slouží *Summary*. *Issue Type* umožňuje rozdělit jednotlivé záznamy (problémy) do kategorií dle jejich významu (chyba, vylepšení apod.). Atribut *Status* označuje stav, ve kterém se nachází daný problém. Kromě stavu se jistě hodí sledovat i prioritu úkolů – *Priority*. *Resolution* určuje, zda byl problém vyřešen, případně je nekompletní nebo například nejde vyřešit.

Dále se nacházejí atributy, které nesou časovou informaci, interagující osoby případně popisy verzí apod. Každý problém musí být nahlášen a přiřazen určité osobě. Atribut *Reporter* eviduje osobu, která zadala úkol, a *Assignee* osobu, které je danému problému přiřazena. Následující trojice atributů *Created*, *Updated* a *Resolved* obsahuje čas a datum, kdy byla položka vytvořena, upravena anebo vyřešena. *Affects Version/s* a *Fix Version* udává, v jakých verzích programu se problém nachází, případně v jaké verzi byl již vyřešen. Pokud s problémem souvisí nějaká komponenta, tak je zachycena v *Components*. *Due Date* vymezuje termín, do kterého by měl být záznam vyřešen. *Votes*, *Watchers* a *Images* se váže spíše na zakomponování hlasování nebo přiřazení obrázků. Jsou zde zaznamenáni uživatelé, kteří mohou dostávat novinky o změnách apod.

Následující atributy se věnují odhadům vynaloženého času na práci a úzce spolu souvisí. *Original Estimate* vyjadřuje původní odhad množství času pro daný problém. *Remaining Estimate* je aktuální odhad zbývajícího času. Evidujeme i čas, jak dlouho na problému pracujeme – *Time Spent*. Nicméně nemusí však nutně platit, že součet zbývajícího a stráveného času musí být roven původnímu odhadu. *Work Ratio* pak udává poměr mezi stráveným časem a původním odhadem.

Pro účely rozmělnění problému na více podúkolů slouží *Sub-Task*, který umožňuje sledovat jednotlivé úkoly odděleně. Následující atributy jsou spíše popisného charakteru a nemusí být vždy vyplněny. Jedná se například o související problémy (*Linked Issues*), prostředí (*Enviroment*), popis problému (*Description*) a omezení přístupu pro uživatele (*Security level*). *Progress* symbolizuje dosavadní průběh a plnění problému.

Nacházejí se zde i celkové sumy některých atributů, protože s postupem času se evidované hodnoty na problému mohou různit. Tím je myšlena například situace, kdy se daný problém považuje za uzavřený a někdy v budoucnu se znovu otevře. Vytvoří se tak nové odhady a čas strávené práce se

vynuluje. Najdeme zde proto sumy celkového průběhu (*Σ Progress*), stráveného času (*Σ Time Spent*), původního odhadu (*Σ Original Estimate*) a zbývajícího odhadu (*Σ Remaining Estimate*).

Poslední atributy nejsou povinné a ani nenesou důležité informace. Za větší zmínku stojí pouze štítek (*Label*) a vlajky (*Flagged*), díky kterým můžeme filtrovat data ještě dalším způsobem. Není problém kromě priority řadit problémy i číselně za sebou dle hodnostní (*Rank/Global Rank*).

Upřesním ještě již zmíněnou trojici velice důležitých a hlavě užitečných atributů – *Issue Type*, *Status* a *Priority*. Popíši zde jejich hodnoty/stavy, kterých mohou nabývat. Kombinace těchto hodnot totiž budeme využívat při třídění dat a následné tvorbě metrik, případně jejich členění.

Typ problému (*Issue Type*):

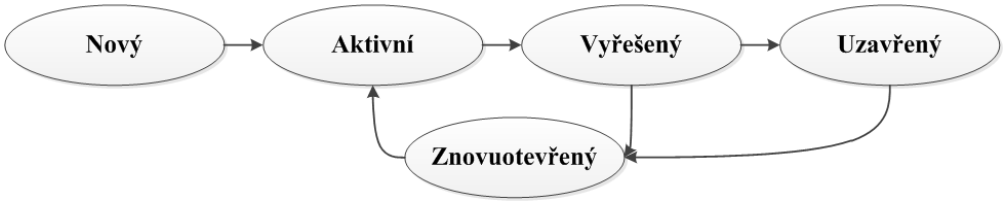
- Chyba (*Bug*)
- Vylepšení (*Improvement*)
- Nová vlastnost (*New Feature*)
- Úkol (*Task*)
- Pod-úkol (*Sub-Task*)
- Technický úkol (*Technical Task*)
- Požadavek (*Request*)
- Příběh (*Story*)

Stav (*Status*):

- Nový (*Open*)
- Aktivní (In Progress)
- Vyřešený (*Resolved*)
- Znovuotevřený (*Reopened*)
- Uzavřený (*Closed*)

Pro lepší pochopení metrik objasním stavy, které mohou sledované položky nabývat (obrázek 17). Také zde zmíním možné přechody mezi jednotlivými stavy sledovaných položek.

Každá položka, která je dána do systému, je označena stavem *nový*. Jakmile je někomu přiřazena ke zpracování a dotyčný na ní začne pracovat, tak se přesune do stavu *aktivní*. Po nějakém čase, kdy je vše na dané položce kompletní se označí stavem *vyřešený*. Nicméně to neznačí, že je vše ověřeno (například zákazníkem), až poté je možný přechod do stavu *uzavřený*. *Znovuotevřený* stav slouží k účelům, kdy se například objeví chyba nebo je požadováno vylepšení dané položky.



Obrázek 17: Stavy sledovaných položek metrik

Priorita (*Priority*):

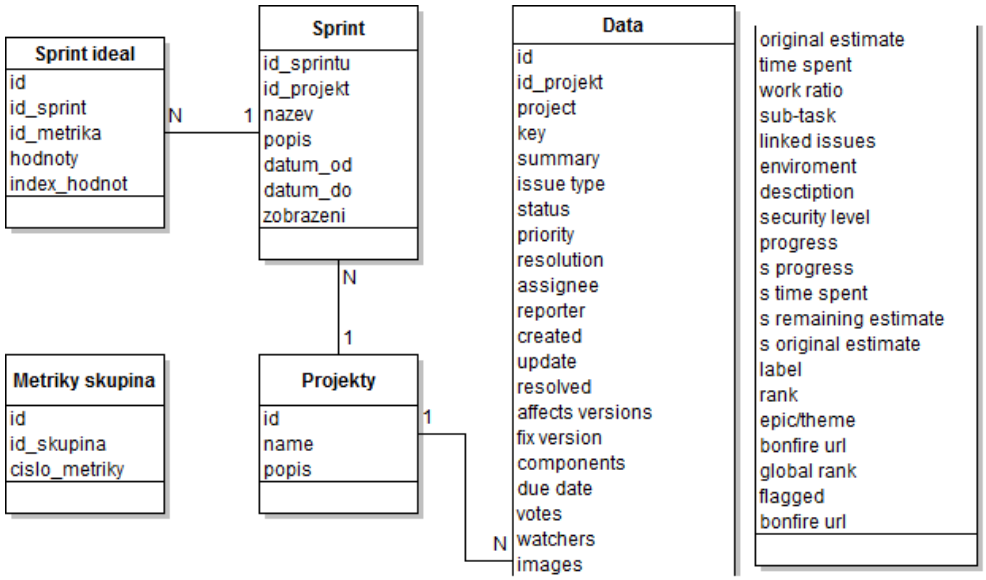
- Triviální (*Trivial*)
- Menší (*Minor*)
- Hlavní (*Major*)
- Kritická (*Critial*)
- Blokátor (*Blocker*)

Kromě typů a stavů problému je také dobré mezi nimi rozlišovat priority, kterými se mohou řídit vývojáři při jejich práci. Méně důležité položky tak mohou odsunout do pozadí a pracovat na kritických problémech, nebo těch, které blokují průběh ostatních.

K našim účelům jsme použili testovací data, která jsou volně dostupná [8]. Počet záznamů čítá okolo 5000, takže by měly mít jistou vypovídající hodnotu. Nicméně by bylo jistě lepší použít data z reálných projektů, která by lépe odrážela situaci.

8.2.2 ER diagram

Na diagramu níže vidíme rozvržení systému na jednotlivé datové tabulky včetně seznamu atributů a vazeb (obrázek 18).



Obrázek 18: ER diagram

Tabulka *Projekty* eviduje všechny projekty, které systém obsahuje. Je v ní uložen unikátní identifikátor projektu (číslo), název a krátký popis. Dále je napojena na další dvojici tabulek – *Data* a *Sprint*.

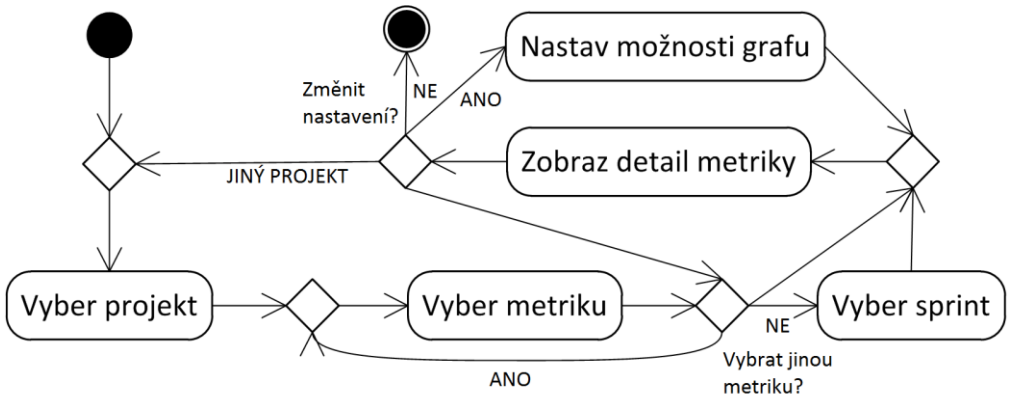
Tabulka *Data* obsahuje datové záznamy z jednotlivých systémů. Její struktura je nakonec shodná s *Jirou* (kapitola 8.2.1) a vzhledem k velkému počtu atributů a celkové robustnosti na ni není problém namapovat data z ostatních systémů [7, JasperForge].

Projekt může obsahovat více časových úseků, které se nazývají sprinty a tvoří obsah tabulky *Sprint*. Eviduje se zde unikátní číslo sprintu a klíč projektu, na který se daný sprint vztahuje. Dále jsou zde atributy pro název a popis, ale nejdůležitější je *datum_od* a *datum_do*, které udávají časové rozmezí sprintu. Poslední atribut značí, zda se bude sprint zobrazovat dle dní (hodnota 0) anebo měsíců (1).

Pro účely měření mohou tyto sprinty mít nastaveny ideální hodnoty, což vyjadřuje tabulka *Sprintideal*, která obsahuje seznam ideálních (očekávaných) hodnot pro jednotlivé sprinty projektu. V grafu jsou pak zobrazeny, jak aktuální hodnoty, tak i ty očekávané. Tabulka opět obsahuje unikátní identifikátor. Je zde i dvojice záznamů pro určení k jakému sprintu a metrice evidovaná položka patří. Očekávané hodnoty jsou ukládány jako posloupnost čísel oddělených čárkou ve stejnojmenném atributu hodnoty. Poslední atribut *index_hodnot* určuje, k čemu se ideální hodnoty váží. Může se jednat o hodnoty pro prioritu (0), stav (1) anebo řešení (2). Díky tomu mohou jednotlivé metriky odrážet více skutečností.

8.3 Diagram aktivit

Diagram aktivit zachycuje situaci, kdy si uživatel chce zobrazit určitou metriku, která náleží do nějakého z projektů (obrázek 19). Níže pod diagramem je zmíněn i popis v textové formě případu užití.



Obrázek 19: Diagram aktivit – zobrazení detailu konkrétní metriky

Případ užití:

1. *Uživatel* vybere jeden projekt z nabídky projektů na úvodní stránce.
2. *Systém* zobrazí detail projektu a metriky, které se k němu vztahují.
3. *Uživatel* vybere jednu z metrik na stránce detailu projektu.
4. *Systém* zobrazí detail metriky
5. *Uživatel* v tomto okamžiku může přejít ke kroku (6 nebo 9), případně úplně ukončit interakci se systémem
6. *Uživatel* vybere sprint
7. *Systém* zobrazí sprint
8. *Uživatel* provede některou z akcí z kroku 5
9. *Uživatel* nastaví možnosti grafu metriky
10. *Systém* zobrazí upravený graf metriky
11. *Uživatel* provede některou z akcí z kroku 5

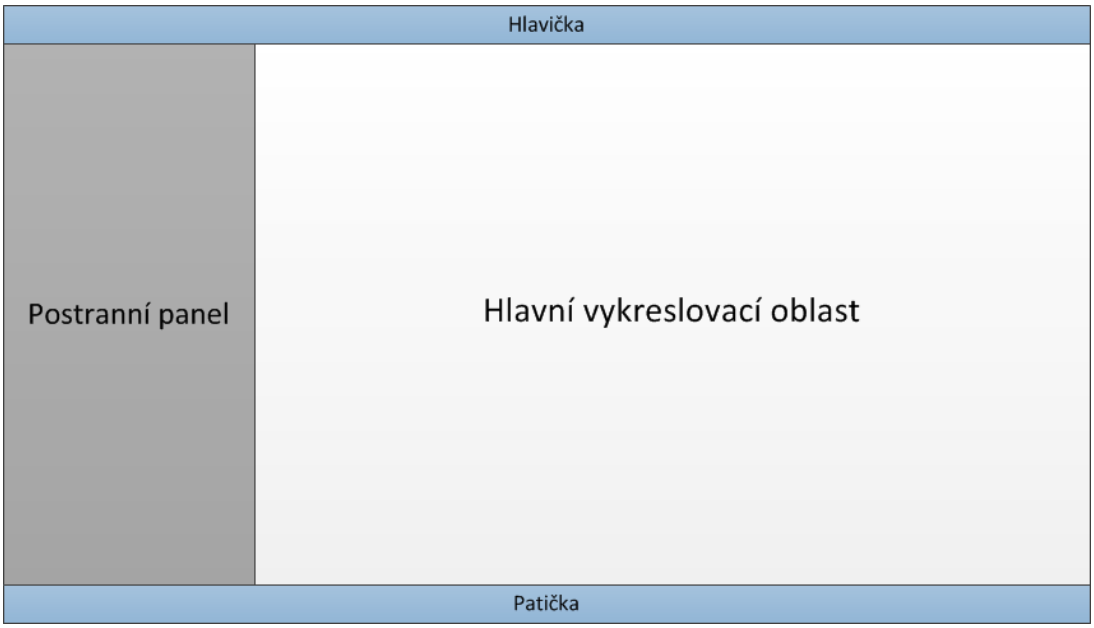
8.4 Návrh uživatelského rozhraní

Při návrhu grafického uživatelského rozhraní (GUI) jsme museli vycházet z faktu, že vyvíjíme webovou aplikaci, která může mít z hlediska uživatelského komfortu jistá omezení (oproti desktopové aplikaci), a je tak třeba s tímto faktem počítat. Řídili jsme se běžnými zásadami, při kterých jsme dbali hlavně na srozumitelnost, přehlednost a jednoduchost použití.

Uživatelské rozhraní je tvořeno celkem čtveřicí základních elementů (obrázek 20). V hlavičce bude zobrazen jen název aplikace, ale mohly by se zde v budoucnu vyskytovat ovládací prvky pro přihlašování a odhlašování uživatelů nebo odkaz pro nápovědu. V patičce bude jen nezbytné minimum – rok dokončení a jména autorů aplikace.

Postranní panel už bude poskytovat větší funkcionalitu a bude sloužit k účelům navigace. Položky, které se v něm budou nacházet, by se daly rozdělit do tří základních skupin. První skupinu tvoří projekty, které budou obsahovat jednotlivé metriky. Zbývající dvojici tvoří správa dat a nastavení.

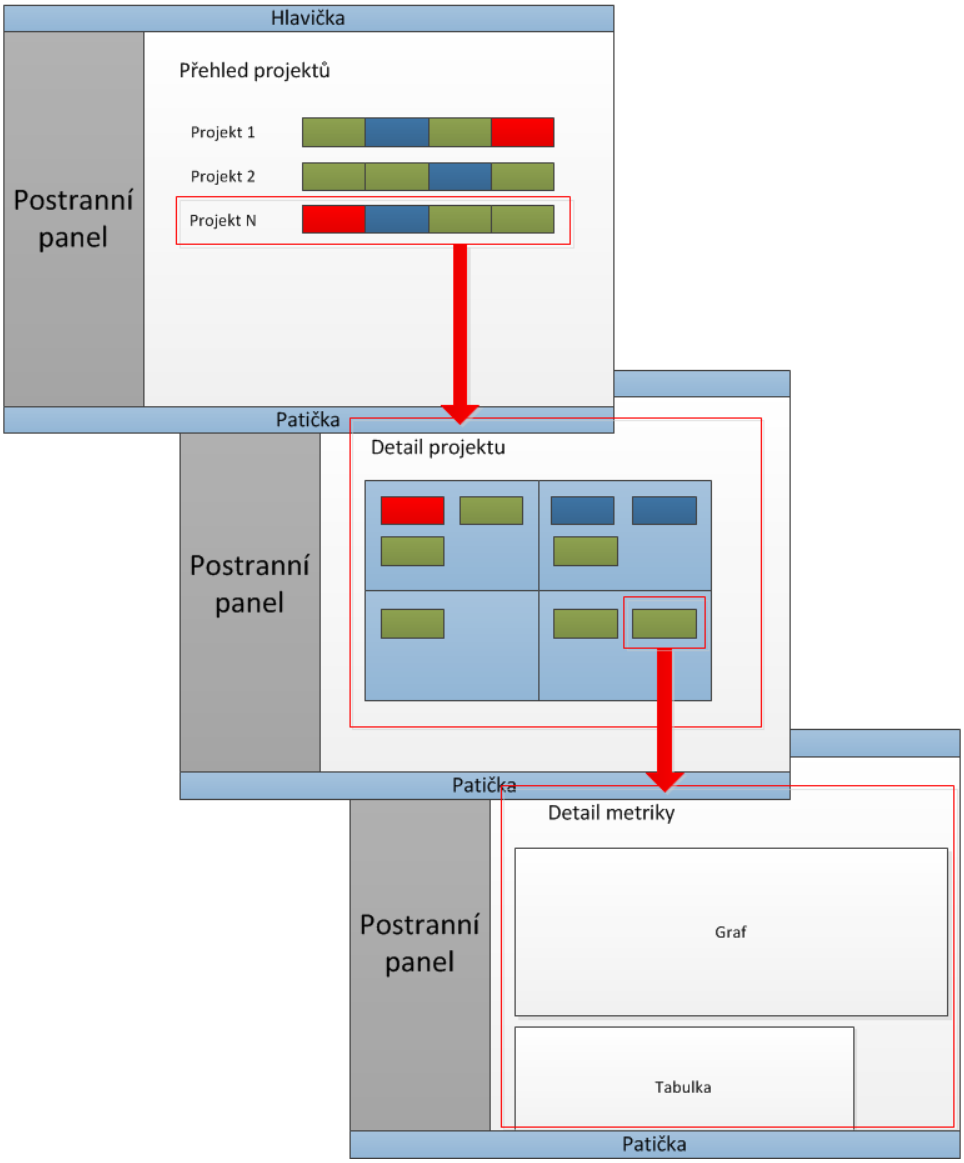
Zbývající a hlavně největší část webové aplikace je vyhrazena zobrazovací oblasti, ve které se budou zobrazovat jednotlivé položky vybrané z navigace. Může se tak jednat o projekty, jejich detaily nebo nastavení metrik, případně import dat. Samotná navigace bude rovněž možná přes hlavní vykreslovací oblast, která bude plně interaktivní a bude obsahovat hypertextové odkazy, stejně jako tomu bude u položek postranního panelu.



Obrázek 20: GUI - Základní rozvržení

V současném návrhu počítáme pouze s jednou uživatelskou rolí a tou je projektový manažer, který bude mít přístup ke všem funkcím webové aplikace. S tím souvisí i plánované rozvržení včetně úvodní stránky apod.

Uživatelské rozhraní by se dalo rozdělit do tří vrstev, pokud bychom se zaměřili pouze na projekty a metriky (obrázek 21). Přehled projektů je první z nich a obsahuje názvy všech projektů v systému. Každý projekt obsahuje celkem čtyři kategorie metrik, čemuž odpovídá čtveřice obdélníků, které mají různé barvy. Tyto barvy značí, zda je v nějaké kategorii problém či nikoliv.



Obrázek 21: GUI – Úrovně projektů a metrik

Na druhou úroveň se dostaneme otevřením detailu projektu z hlavní vykreslovací oblasti nebo postranního panelu. Vidíme už jednotlivé metriky i do kterých kategorií spadají. Opět jsou barevně rozlišeny pro lepší přehled.

Po otevření některé z metrik se dostaneme na její konkrétní detail, kde najdeme graf, tabulku, textové popisky. Bude zde možné nastavit filtry, podle kterých se bude metrika zobrazovat.

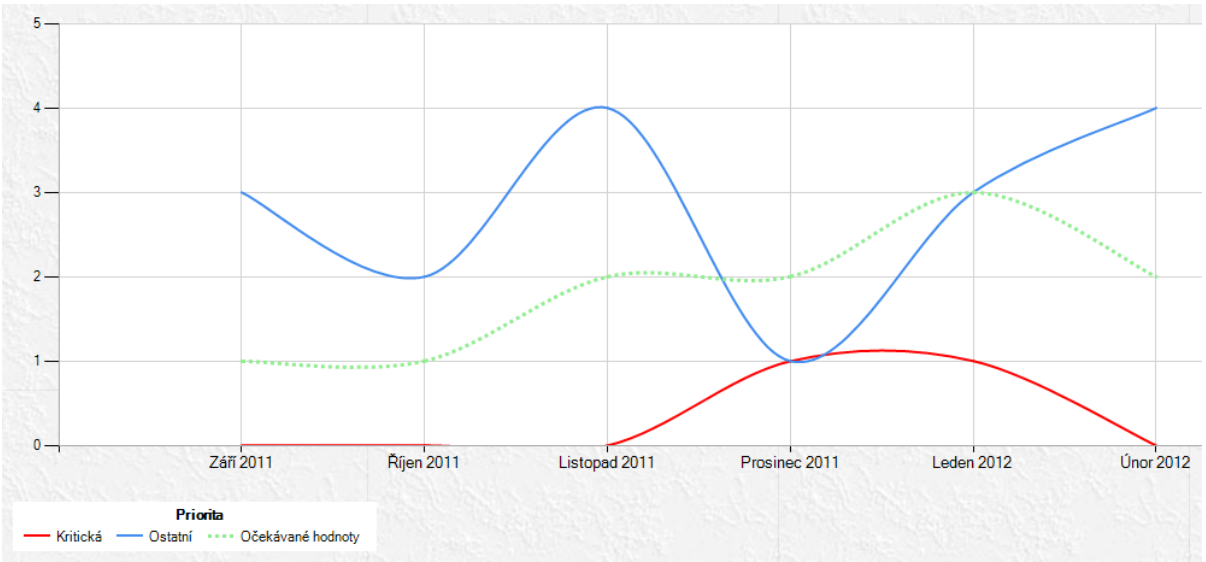
9 Návrh metrik

V následující kapitole se budu věnovat návrhu a popisu metrik pro měření kvality softwarového vývoje. Jednotlivé metriky byly navrženy a vytvořeny na základě dostupných dat, kde hlavní myšlenka spočívala v rozdělení záznamů dle typu problému. Tímto problémem jsou myšleny například chyby, požadavky, úkoly apod.

Vzniklé metriky tak obsahují počty položek za jednotlivé dny, případně měsíce a po vzoru metody *Scrum* jsou rozděleny na sprinty. Pro každý sprint je možné nastavit trend, což není nic jiného než skupina očekávaných hodnot, které se při vyhodnocení metriky porovnají s těmi aktuálními. Můžeme tak ihned vidět rozdíly mezi plány a realitou a díky tomu nastolit nápravná a preventivní opatření.

9.1 Požadavky

Pouze v malém počtu situací jsou uživatelé plně spokojeni s funkcionalitou softwaru. Kladou tak často nové požadavky, které se týkají funkcí softwaru. Může se jednat o zcela novou funkci nebo pouhé vylepšení té stávající. Následující dvojice metrik se bude věnovat právě prvnímu z případů, konkrétně vyřešeným a uzavřeným požadavkům.



Obrázek 22: webová aplikace – vyřešené požadavky

9.1.1 Vyřešené požadavky

Metrika *vyřešených požadavků* udává počet požadavků, které jsme v projektu vyřešili za určité období dle jejich priority. Jedná se o požadavky, které jsme dostali zadány a už jsme je byli schopni vyřídit. Vysoká míra vyřešených požadavků je indikátorem, který vyjadřuje, že tým vývojářů pracuje dobře. Vždy bychom však měli porovnat aktuální počet aktivních požadavků s těmi vyřešenými. Může totiž nastat situace, že počet vyřešených požadavků bude nulový a můžeme tak nabýt dojmu, že

tým pracuje špatně. Záleží tedy na výše zmíněných okolnostech a také na faktu, že během vývojového cyklu softwarového díla se charakteristika požadavků bude měnit.

V grafu výše (obrázek 22) jsou zaznamenány kritické a ostatní požadavky. Kritickým přísluší červená barva, protože mají vyšší prioritu a jsou pro tým důležitější. Do ostatních (modrá barva) spadají problémy s triviální, malou anebo hlavní prioritou. Z grafu je možné vyčíst i ideální hodnoty, které reprezentuje zelená tečkovaná křivka. Můžeme tak porovnat aktuální data oproti očekávaným. V našem případě řeší tým vývojářů požadavky nad očekávání. V prosinci se může zdát, že bylo očekávání vyšší, než počet vyřešených požadavků. Je ale třeba mít na paměti, že se požadavky dělí dle priority a musí se brát jako součet, který je v tomto případě shodný s očekávanými hodnotami.

9.1.2 Uzavřené požadavky

Následující metrika *uzavřené požadavky* opět pracuje s požadavky. Už jsem zmínil výše, že existují nové a vyřešené požadavky a co tyto stavy reprezentují. Vyřešený požadavek není konečným stavem, ale znamená, že byla týmem vývojářů zahrnuta požadovaná funkcionalita do implementace. Neznamená to ale, že splňuje uživatelem kladené nároky. Jedině pokud vyřešený požadavek projde uživatelskou inspekcí a vyhoví, tak se může uzavřít. Takové případy bude s mírnou odlišností řešit právě metrika nazvaná *uzavřené požadavky*.

V ideálním případě tak budou ubývat vyřešené požadavky, protože budou přecházet do uzavřeného stavu. Je vhodné zmínit, že uzavření není zcela definitivní. Může se stát, že uživatel přehlédl nějakou chybu, kterou je nutné opět řešit, a tak se požadavek opět otevře. Ne vždy tak uzavřené požadavky musí znamenat, že je daný požadavek definitivně zpracován.

9.2 Uzavřené vylepšení

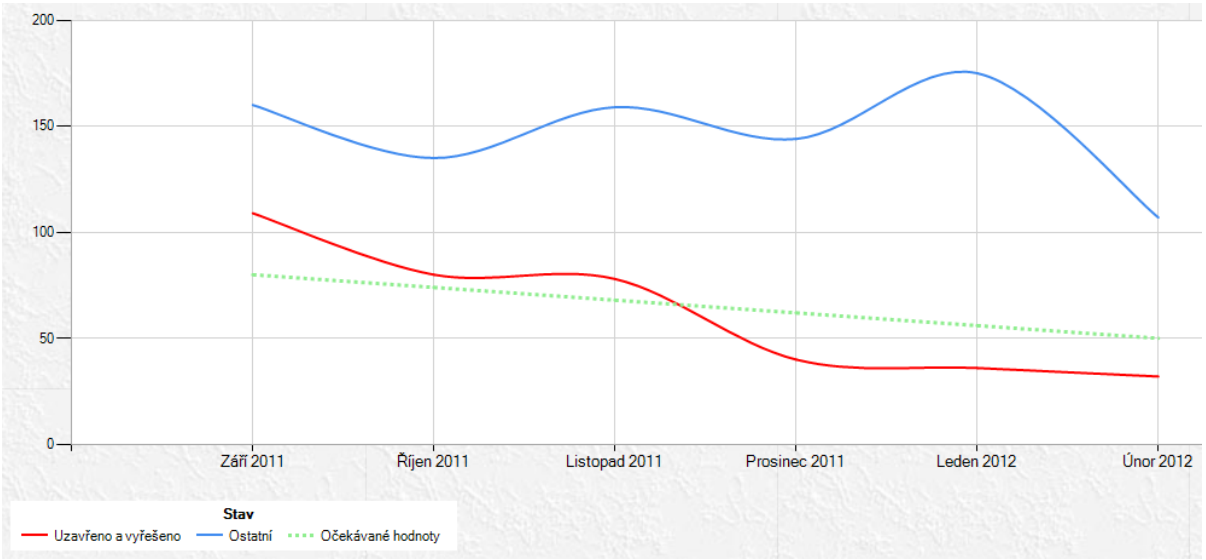
Rozdíl mezi vylepšením a požadavkem jsem již nastínil výše. U vylepšení jde pouze o úpravu stávajících vlastností a funkcí. Může se jednat například o vizuální změny nebo zvýšení efektivity výpočtu apod. Nejedná se tedy o přidání dalších nových funkcí.

Metrika *uzavřených vylepšení* je analogicky podobná *uzavřeným požadavkům*, kde hlavním rozdílem je typ evidovaného problému. V ideálním případě budou uzavřená vylepšení převažovat a pouze malá část jich bude přibývat anebo bude aktuálně řešena. Pokud tým vývojářů není schopen uzavírat vylepšení a stále přibývají další a další požadavky, je zřejmé, že by měl projektový manažer uvážit navýšení počtu členů nebo dokonce přidělit tyto problémy jinému týmu.

9.3 Uzavřené chyby

Přestože se tým vývojářů snaží sebevíce, stejně vznikají chyby. Dodržováním standardů a využitím nejnovějších metod vývoje softwaru se procento výskytu chyb dá zmírnit, ale nikoliv úplně eliminovat. Každá vzniklá chyba tak ovlivňuje nebo dokonce znemožňuje fungování softwaru. Je žádoucí tyto chyby v co nejkratším čase odstranit, aby se zajistila co nejlepší kvalita vyvíjeného softwaru.

Metrika uzavřených chyb vyjadřuje, jak je tým vývojářů schopen řešit chyby. Porovnávají se zde uzavřené a vyřešené chyby oproti těm ostatním (nové, aktivní, znovuotevřené). Tento poměr určuje, jak je schopen tým vývojářů zpracovávat, řešit a hlavně opravovat chyby v softwaru.



Obrázek 23: webová aplikace – uzavřené chyby

Z grafu lze vyvodit hned několik závěrů (obrázek 23). V celkovém měřítku tým vývojářů pracoval dle očekávání. V první polovině dokonce nad očekávání, ale pak nastal okolo listopadu zlom. Může to značit například odchod členů z týmu, nebo že dostali přidělený další projekt apod. I přes poměrně přesný odhad řešení a uzavírání chyb je zde stále přítomno velké procento chyb nových a aktivních. To může být způsobeno například méně kvalitním návrhem, případně špatným naprogramováním.

9.4 Příběhy

Jedná se o neformální vyjádření požadavků pomocí běžného jazyka. Jde se o rychlý způsob nakládání s požadavky zákazníka, bez nutnosti vytváření formalizovaných požadavků a provádění administrativních úkolů s tím spojených. Uživatelské příběhy jsou používány v agilních metodikách vývoje software jako podklad pro definování funkcí systémů.

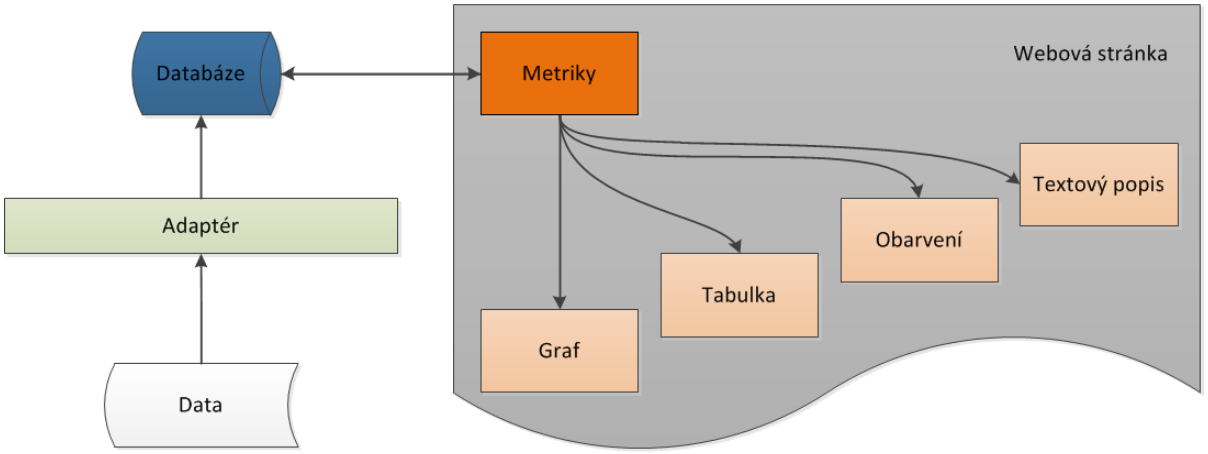
Metrika příběhů nám udává jakýsi vyšší pohled na vytvářený systém a to především na to, jak dokonale je nově vznikající produkt popsán. Především v prvních stádiích vývoje by těchto „příběhů“ mělo být vytvořeno velké množství. Následně by pak jejich počet už neměl nijak výrazně stoupat. To by naznačovalo nedostatečnou analýzu problémů, která by mohla mít jako následek nemalé problémy.

10 Implementace webové aplikace

Následující kapitola je věnována implementaci webové aplikace *Testing reports* vyvíjené v rámci diplomové práce. Webová aplikace je založena na platformě *.NET* od Microsoftu a využívá programovacího jazyka *C#*. Pro uložení dat využíváme databázi Microsoft SQL Server 2008.

10.1 Architektura aplikace

Před popisem základního rozvržení aplikace do tříd a vysvětlením jakým způsobem jsou metriky vypočítávány a hodnoceny, popíši architekturu spíše abstraktním způsobem. Už z návrhu uživatelského rozhraní se dalo vytušit jisté rozdělení do úrovní a základní funkcionalita apod. Architektura tomuto původnímu návrhu z velké části odpovídá.



Obrázek 24: Architektura I.

Následující schéma velmi zjednodušeně znázorňuje jádro aplikace (obrázek 24). Před vyhodnocením dat musí předcházet jejich načtení skrze adaptér do databáze. Data jsou zde transformována do jednotné struktury, díky které můžeme jednotlivé projekty (i z odlišných systémů) vyhodnocovat stejnými postupy. Modul metrik má na starosti napojení se do databáze a následné vyhodnocení metriky. Výstupem je jednak graf a tabulka, tak obarvení, ke kterému se dostanu později, anebo textový popis.

Samotné jádro by pro funkci webové aplikace nepostačovalo. Vyskytují se zde i další komponenty, které poskytují další funkce. Jedním z hlavních požadavků byla agregace dat z více rozdílných systémů. Je tak zřejmé, že je nutné pro jednotlivé projekty vytvořit souhrnné přehledy, jinak by v datech vznikl chaos. S tím souvisí právě druhé podrobnější schéma, které využívá rozdělení projektů a metrik na více vrstev (obrázek 25).

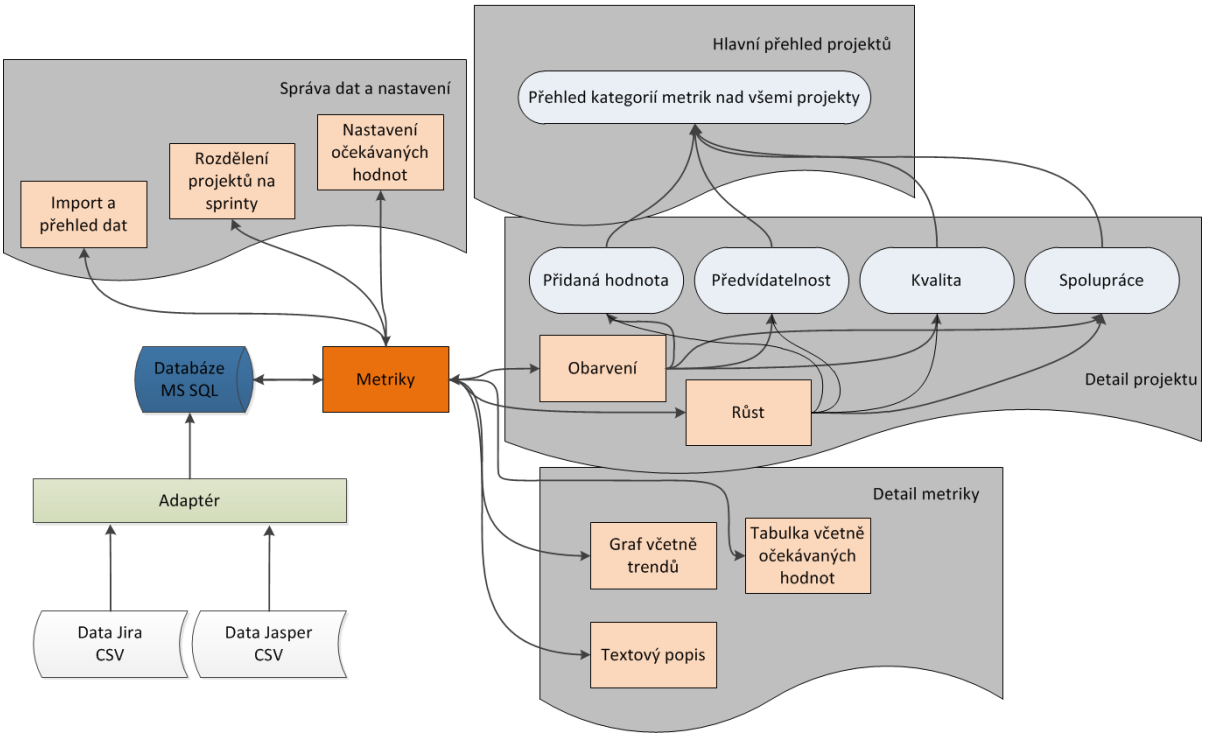
Před popisem úrovní webové aplikace ještě zmíním *import dat*, protože bez nich by vše postrádalo smysl. Data z jednotlivých projektů se musí nejprve přetransformovat ze systému *JIRA* a *JasperForge* na naši interně používanou strukturu a následně uložit do databáze. Získáme tak skupinu projektů s unikátními názvy, u kterých můžeme v *přehledu dat* nahlížet na jejich jednotlivé

záznamy. Dalším krokem je *rozdělení projektů na sprinty*, bez kterých by nešlo nahlížet na projekty v jednotlivých metrikách.

Teď už nic nebrání přechodu k *detailu metriky* zobrazující nejnižší úroveň, kde se nachází konkrétní metrika. Za aktuální situace by se zobrazila data z projektů u všech metrik. Nicméně je potřebné u každé metriky ještě *nastavit očekávané hodnoty*. Ty se pak porovnávají s aktuálními daty a výsledkem je *barevné ohodnocení* nebo *růst* dané metriky.

O vrstvu výše se nachází detail projektu. Obsahuje následující kategorie metrik – přidaná hodnota, předvídatelnost, kvalita a spolupráce. Každá metrika nese kromě svého názvu také barevnou informaci a šipku, kterou dědí z nižší úrovně. *Barva* vyjadřuje kvalitu plnění metriky a *růst* změnu dat oproti poslednímu měsíci. Podrobněji popíši tyto hodnotící kritéria v textu dále.

Na nejvyšší úrovni je zobrazen *hlavní přehled projektů* obsahující všechny projekty, které jsou naimportovány v systému. Vedle každého projektu je zobrazena čtveřice výše zmíněných kategorií metrik, které opět nesou barevnou informaci stejně, jako tomu bylo u dílčích metrik. Jediný rozdíl je ten, že barva zde platí pro celou kategorii.



Obrázek 25: Architektura II.

10.2 Logika obarvení

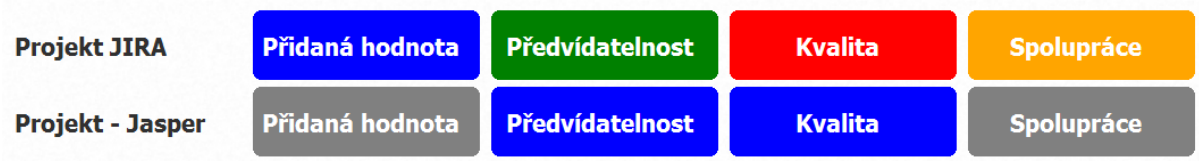
Vzhledem k rozlišovacím schopnostem lidského oka a fungováním mozku samotného se myšlenka využít barvy pro hodnocení jednotlivých metrik nebo jejich kategorií přímo vybízela. Pro každý sprint metriky se vypočítá procentuální odchýlení aktuálních hodnot od původního odhadu. Sprint, který dopadl nejhůře, bude reprezentovat danou metriku (obrázek 26a). Jeho barva se určí z hodnotící tabulky (obrázek 26b), kde jsou rozsahy odchýlení včetně obarvení. Ne vždy musí nutně

znamenat velké vychýlení problém, protože tým může například řešit problémy rychleji, než se očekávalo. Tímto způsobem se obarvení aplikuje na všechny metriky.

Procenta	Barva
0-69%	Zelená
70-89%	Tmavě zelená
90-110%	Modrá
111-130%	Oranžová
131% a více	Červená
Chybějící data	Šedá
Nulové hodnoty reálných hodnot	Hnědá

Obrázek 26a/b: Obarvení metriky, hodnotící tabulka

Barevnou informaci nesou i kategorie jednotlivých projektů (obrázek 27). Výsledné obarvení vždy odpovídá nejhorší metrice z dané kategorie. Původně jsme zamýšleli jednotlivým metrikám přiřadit váhy, ale tento nápad se ukázal jako ne zcela ideální. Může totiž nastat situace, ve které většina metrik vyjde lépe, než jsme očekávali a pouze jedna jediná na tom bude velice špatně. S využitím vah by se tato skutečnost vůbec nemusela projevit v nadřazené kategorii metrik a realita by byla značně zkreslená.



Obrázek 27: Obarvení kategorie projektů

10.3 Logika růstu

Barevná reprezentace metrik a jejich jednotlivých sprintů není jediným hodnoticím kritériem. Výpočet totiž probíhá nad všemi sprinty odděleně a není v něm zohledněna jejich vzájemná návaznost. Bylo by jistě dobré mít přehled o změnách problémů týkajících se každé z metrik mezi jednotlivými sprinty.

Obrázek 28: webová aplikace – růst

Šipka u každé metriky v detailu projektu značí změnu problémů (růst) mezi posledním a předposledním sprintem (obrázek 28). Pokud je například počet chyb oproti minulému měsíci větší, tak šipka směřuje směrem vzhůru. V opačném případě bude směřovat dolů, anebo pokud se budou problémy držet v určitém procentuálním rozsahu, tak bude šipka ve vodorovném směru, který značí

stabilní stav. Z obrázku lze tak vyčíst, kolik uzavřených úkolů oproti minulému sprintu přibylo a kolik uzavřených chyb se drželo v rovnovážné hladině.

Výpočet se nezakládá na pouhém porovnání sum problémů, ale přidali jsme do něj ještě váhy. Ty zohledňují priority jednotlivých problémů, díky kterým získáme přehled více odpovídající realitě. Problémy s vysokou prioritou totiž vyžadují větší vynaložené úsilí a naopak.

10.4 Popis funkcí tříd

V předchozích podkapitolách jsem se věnoval abstraktnímu náhledu na architekturu webové aplikace a hlavně logice výpočtů metrik. V této podkapitole bych chtěl nastínit už fyzickou strukturu aplikace, která je rozdělena více do tříd. Tyto třídy se starají například o správu a výměnu dat, výpočty, nebo o zobrazení webové stránky.

10.4.1 default

Třída *default* je základním stavebním prvkem celé aplikace. Generuje většinu html kódu a stará se o zobrazení dílčích částí a prvků webové aplikace. Jedná se hlavně o zobrazení přehledu projektů na úvodní straně a také o detailnější přehled projektů v nižší úrovni. Pro tyto účely se volají instance třídy metrics_class. Tyto instance vrací data konkrétní metriky, která jsou následně zpracována a vyhodnocena. Jedná se hlavně o obarvení metriky a zobrazení příslušné šipky.

10.4.2 metrics a metrics_class

Kromě obarvení a růstu se na nejnižší úrovni u každé metriky zobrazuje i datová tabulka a graf. Toto má na starosti právě třída *metrics*. Opět přes instanci třídy *metrics_class* se získají informace a data pro konkrétní metriku, která se následně zobrazí v datové tabulce. Třída *metrics_class* se připojuje do MS SQL databáze a zajišťuje předání požadovaných výstupů třídě *metrics*. Se stejnými daty pracuje i třída *graph_class*, která má na starost výstup grafu.

10.4.3 graph_class

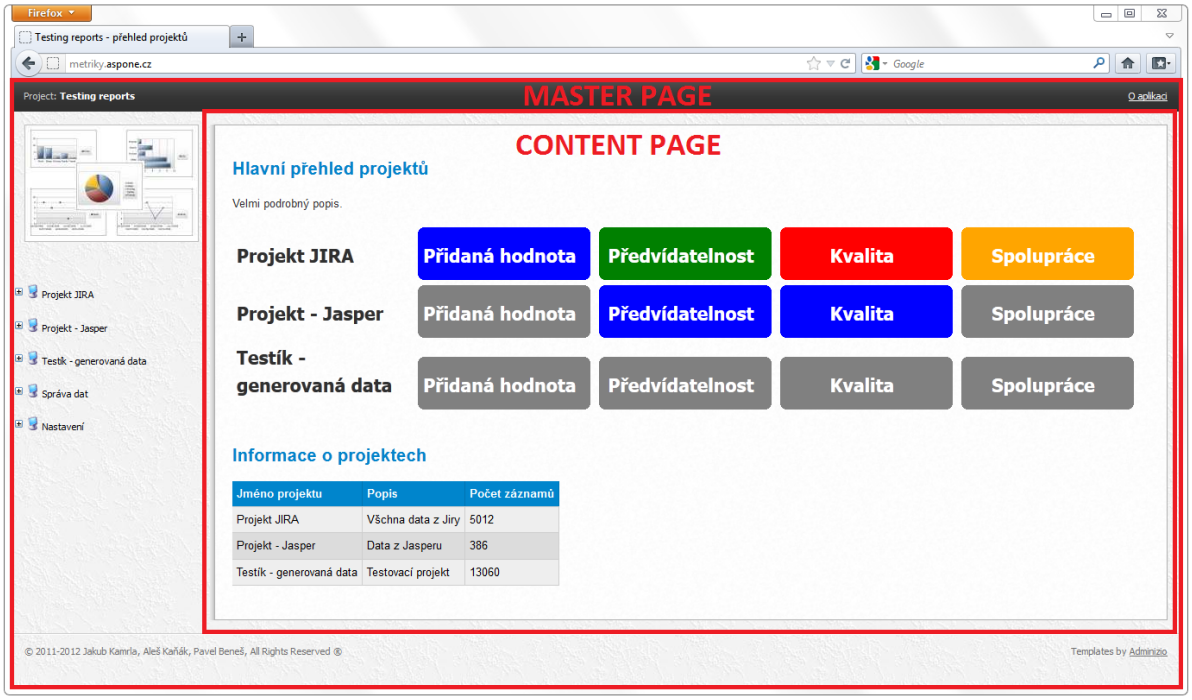
Třída *graph_class* zajišťuje předpřipravení dat, které následně zobrazuje komponenta *Chart*. Hlavním účelem třídy je tak přetransformovat vstup ve formě informací a dat, který bude odpovídat požadované struktuře, se kterou dokáže komponenta Chart pracovat. Jedná se o datovou tabulku (DataTable), díky níž se mohou vykreslit data v grafu. Dále je zde možno specifikovat typ grafu a další možné filtry.

10.4.4 data

Nejenom import a přehled dat zajišťuje třída *data*. Obsahuje kromě toho i nastavení očekávaných hodnot, přiřazení metrik do skupin a rozdělení projektů na sprinty. Data na základě požadavků uživatele čte nebo zapisuje z databáze.

10.5 Uživatelské rozhraní

S uživatelským rozhraním, které vychází z návrhu architektury, úzce souvisí speciální třída zvaná *Site.master*, která umožňuje velice jednoduchý vývoj a následné úpravy webových stránek. Obsahuje šablonu – *Master Page*, která určuje jednotné rozvržení a vzhled celé webové aplikace. *Master Page* obsahuje dva základní typy oblastí. První z nich je společná pro všechny stránky. Jedná se ve většině případů o použití hlavičky, patičky a menu, případně loga aplikace. Druhá oblast je už unikátní pro každou stránku (*Content Page*), kde se vkládá její zdrojový kód. Díky těmto skutečnostem je vyvíjení a hlavně následná úprava stránek velice jednoduchá a nedochází k nekonzistenci údajů.



Obrázek 29: webová aplikace – úvodní stránka

V našem konkrétním případě jsme využili pouze jednu *Master Page*. Není problém těchto typů stránek využít více a různě je do sebe zanořovat. *Master Page* obsahuje hlavičku, patičku a levý postranní panel. V hlavičce se nachází pouze název a informace o aplikaci. V patičce najdeme pouze jména autorů. Levý postranní panel obsahuje kromě loga aplikace také navigační menu, které umožňuje pohyb mezi jednotlivými webovými stránkami. Část, která obsahuje projekty včetně metrik, je generována automaticky na základě dat z databáze. Správa dat a nastavení je zde nastavena na pevno, staticky. K vykreslení navigace využíváme instanci třídy *TreeView*.

10.6 Ukázky implementace

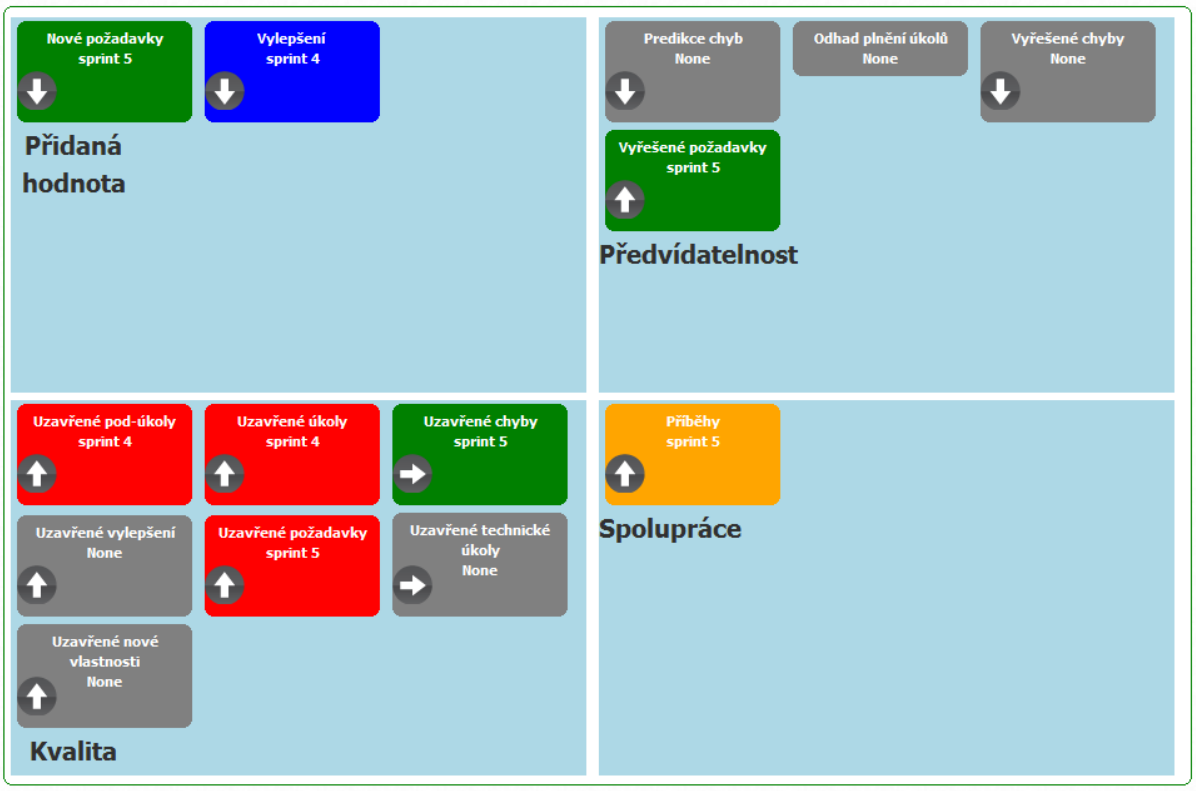
Následující podkapitola je věnovánu ukázkám webové aplikace, která je dostupná na internetové adrese http://metriky.aspone.cz [9]. Využili jsme služeb *free webhostingu ASPone*, který má však svá úskalí a omezení. Jedná se například o velikost prostoru pro data a databázi. Rychlost

zpracování požadavků není zcela ideální, ale pro účely demonstrace funkcí webové aplikace bude postačovat. Vývoj a testování probíhalo na lokálním serveru.

V předchozích dvou kapitolách jsem již představil jednotlivé fragmenty webové aplikace. Minulá podkapitola dokonce ukazovala celkový vzhled úvodní strany. Pokračoval bych v tomto popisu dále a okomentoval stručně vybrané obrazovky.

10.6.1 Detail projektu

K detailu konkrétního projektu se dostaneme z úvodní stránky, která obsahuje přehled všech projektů (obrázek 29) včetně jejich čtyř kategorií metrik. Ty jsou barevně rozlišeny dle hodnotících tabulek a upozorňují na možný zdroj problému. Po otevření některého z projektů se dostaneme na *detail projektu* (obrázek 30). Každá kategorie metrik zde již obsahuje metriky, které jí přísluší. Například v kategorii zabývající se přidanou hodnotou najdeme metriky nových požadavků a vylepšení. Ty jsou barevně ohodnoceny a obsahují také informaci o růstu. Nejhorší metrika z dané kategorie ovlivňuje barevnou informaci o úroveň výše v přehledu projektů. Například kategorie přidané hodnoty bude v přehledu projektů modrá a kvalita bude obarvena červeně. Projektový manažer díky tomu nemusí procházet jednotlivé kategorie metrik a pouze se může věnovat těm, které hlásí možný výskyt problému.



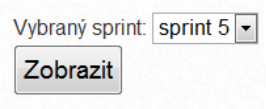
Obrázek 30: webová aplikace – detail projektu

10.6.2 Detail metriky

Z přehledu projektů nebo z jejich detailu se dozvíme základní informace o tom, zda metriky sledující určitý typ problému na základě barvy nebo šipek jsou v pořádku či nikoliv. Nicméně tyto

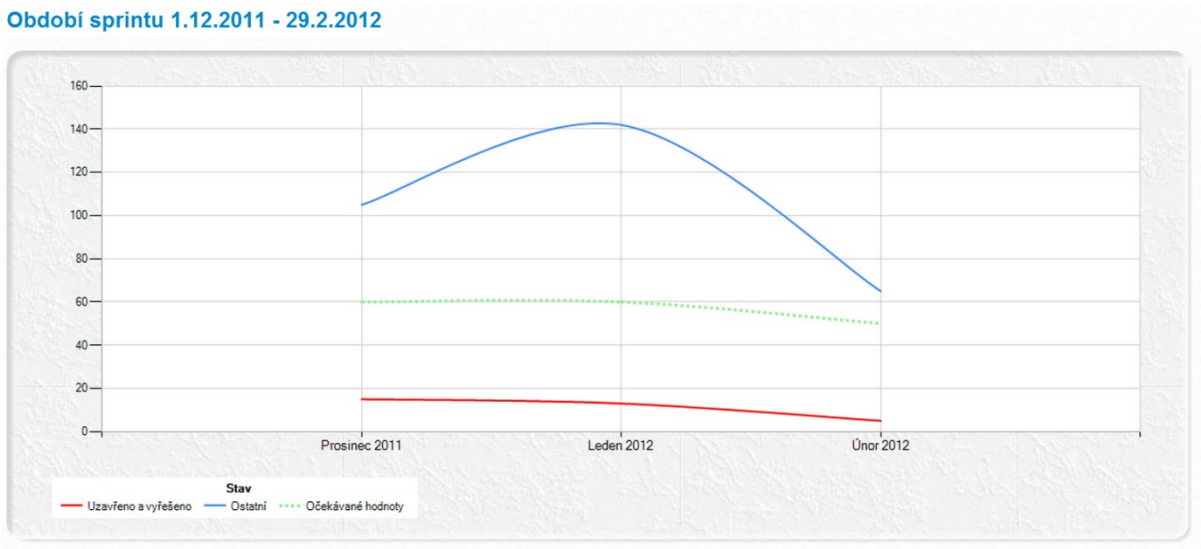
informace jsou pouhým vodítkem, které může odhalit problém. Neznáme zde konkrétní hodnoty a dílčí výsledky jednotlivých metrik.

Z detailu projektu se tak dá přejít na detail jednotlivých metrik, který už obsahuje konkrétní data ve více interpretacích. Vzhledem k tomu, že se projekty mohou skládat z více sprintů, tak i při vyhodnocení metrik můžeme zobrazovat jednotlivé sprinty odděleně (obrázek 31).



Obrázek 31: webová aplikace – detail metriky (sprint)

Data každé metriky jsou vizualizována ve formě grafu a tabulky. O těchto grafech jsem již psal podrobně v návrhu metrik (kapitola 9). Každý tento graf obsahuje naměřená data ze systémů včetně odhadu očekávaných hodnot. Dle nastavení sprintu se zobrazuje podle měsíců nebo dnů. Metrika je tak přehledně zobrazena a projektový manažer může sledovat odchýlení aktuálních hodnot od těch plánovaných.



Obrázek 32: webová aplikace – detail metriky (graf)

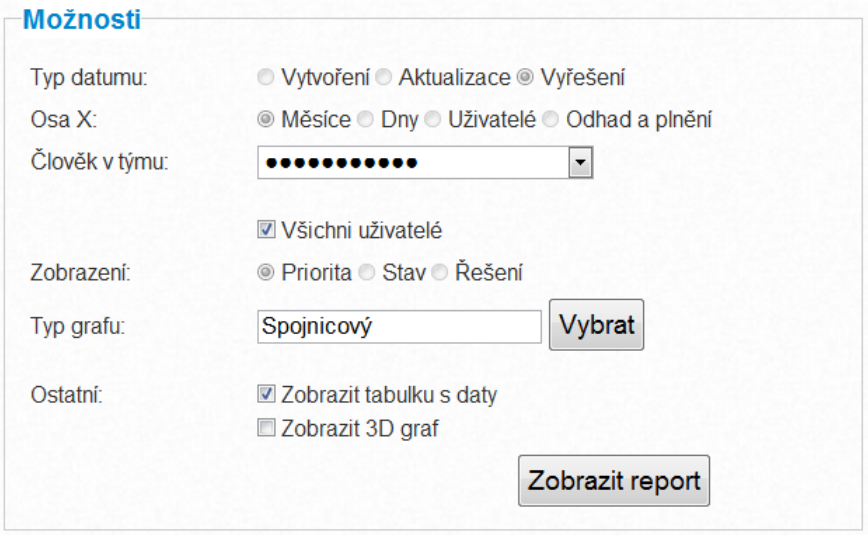
Nedílnou součástí detailu metriky je datová tabulka (obrázek 33). Může dobře posloužit tam, kde graf selže. Tedy v situacích, kdy potřebujeme vědět přesné hodnoty. Jsou zde zobrazena jednotlivá období každého sprintu (dny, měsíce) a počty problémů, které jsou děleny podle stavu, případně priority (závisí dle metriky). Na základě datové tabulky je sestrojen graf, který by nemohl být bez těchto informací zobrazen.

Datová tabulka

Období	Uzavřeno a vyřešeno	Ostatní	Očekávané hodnoty
Prosinec 2011	15	105	60
Leden 2012	13	142	60
Únor 2012	5	65	50

Obrázek 33: webová aplikace – detail metriky (tabulka)

Výstup grafu lze v jistých ohledech upravovat pomocí filtrů (obrázek 34). V základním nastavení se využívá spojnicového grafu, ale je ho možné změnit za jiný typ (například výsečový nebo pruhový). Dále je možné i sledovat jednotlivé členy týmu odděleně a porovnávat tak jejich výkonnost apod. Některé metriky mají i možnost měnit typ zobrazení na základě priority, stavu nebo řešení. Další ukázky obrazovek webové aplikace včetně jejich stručného popisu se vyskytují v příloze B.



Obrázek 34: webová aplikace – detail metriky (filtr)

11 Závěr

Cílem mé diplomové práce bylo nastudovat problematiku zabývající se měřením kvality softwarového vývoje a následně vytvořit *aplikační framework*, který by sledoval celý proces vývoje softwarového díla od zadání projektu až po jeho ukončení. Diplomová práce byla určena pro více řešitelů, se kterými jsem spolupracoval a vyměňoval si s nimi zkušenosti a nápady, díky kterým jsme mohli společně zrealizovat *aplikační framework*.

Zhotovení *aplikačního frameworku* předcházelo důkladné prozkoumání problematiky metrik a vývoje softwaru. Zaměřil jsem se v této části hlavně na odhady vynaloženého úsilí a s tím souvisejícími odhady velikosti softwaru. Věnoval jsem se také nejběžněji užívaným agilním metodám vývoje softwaru a zaměřil se hlavně na metodu *Scrum*. Dále jsem společně s kolegy zmapoval aktuální situaci na trhu s nástroji, které měří kvalitu vývoje softwaru. Díky tomu jsme získali základní povědomí o schopnostech těchto nástrojů. Ukázalo se, že nástroje se mezi sebou diametrálně liší jak kvalitou, tak vizualizací výstupů. Projektový manažer, který pracuje s více systémy, by jistě ocenil unifikované rozhraní pro správu všech projektů na jednom místě.

Aplikační framework byl vytvořen na základě požadavků diplomové práce a obsahuje všechnu požadovanou funkčnost. Při jeho tvorbě jsme vycházeli z dostupných nástrojů na trhu a hlavně z dat, které tyto nástroje evidují. Prvotně jsme zamýšleli vytvořit desktopovou aplikaci, ale tento směr se nakonec neukázal jako ideální. Webová aplikace byla lepší volbou, ta je dostupná téměř odkudkoliv a bez nutnosti instalace. Obsahuje možnost sledovat kvalitu vývoje softwaru více projektů z rozdílných systémů (*JIRA*, *JasperForge*). Při tvorbě metrik jsme dbali na vizuální přehlednost a srozumitelnost. Využili jsme například metodu *Scrum* pro dělení projektů na sprinty nebo základní myšlenku metody funkčních bodů pro nastavení vah u počítání růstu metrik. Směr dalšího vývoje webové aplikace by mohl spočívat v zakomponování dalších systémů (např. *TFS*, *Sprintometer*), přidáním dalších metrik nebo například interaktivních grafů, ve kterých by bylo možné ideální hodnoty nastavovat pouhým tažením myši.

Už na samotném počátku studia problematiky jsem si uvědomoval, že měření kvality vývoje softwarového díla je velmi důležitý proces. Díky němu jsme schopni mít nad projektem kontrolu a dokážeme tak odhalit problémy už v jejich rané fázi. Stejně tak bez měření kvality nemůžeme nijak stanovit, zda jsme se v určitém období zlepšili či nikoliv. Měření kvality vývoje softwarového díla velkou měrou přispívá na dodání produktu vysoké kvality, u kterého se dodrží časový plán a nepřekročí finanční rozpočet. Cílem našeho snažení by měl být spokojený zákazník, který obdrží produkt vysoké kvality splňující všechny jeho představy a kladené požadavky.

Literatura

[1] Erbert, C., Dumke, R., Bundschuh, M., Schmiedendorf, A., *Best Practices in Software Measurement*, Heidelberg, New York, 2004.

[2] Souhrn dostupných metod a standardů vývoje softwaru. Dostupné z URL <http://www.scrumalliance.org/articles/100-agile-and-cmmi-better-together>

[3] Scrum. Dostupné z URL <http://en.wikipedia.org/wiki/Scrum_(development)>

[4] TFS ilustrace. Dostupné z URL < http://www.vss2tfs.net/tfsintro.htm > a < http://www.daquas.cz/articles/402-visual-studio-team-foundation-server-2010>

[5] Sprintometer ilustrace. Dostupné z URL <http://scrumfr.free.fr/sprintometer-305/>

[6] MSDN – TFS reporty. Dostupné z URL < http://msdn.microsoft.com/cs-cz/library/dd380714.aspx>

[7] Kamrla, J., *Diplomová práce*, Ostrava, 2012

[8] JIRA testovací data. Dostupné z URL <https://jira.atlashost.eu/sandbox/browse/JIRA>

[9] Webhosting ASPone. Dostupné z URL < http://www.aspone.cz/>

Seznam příloh

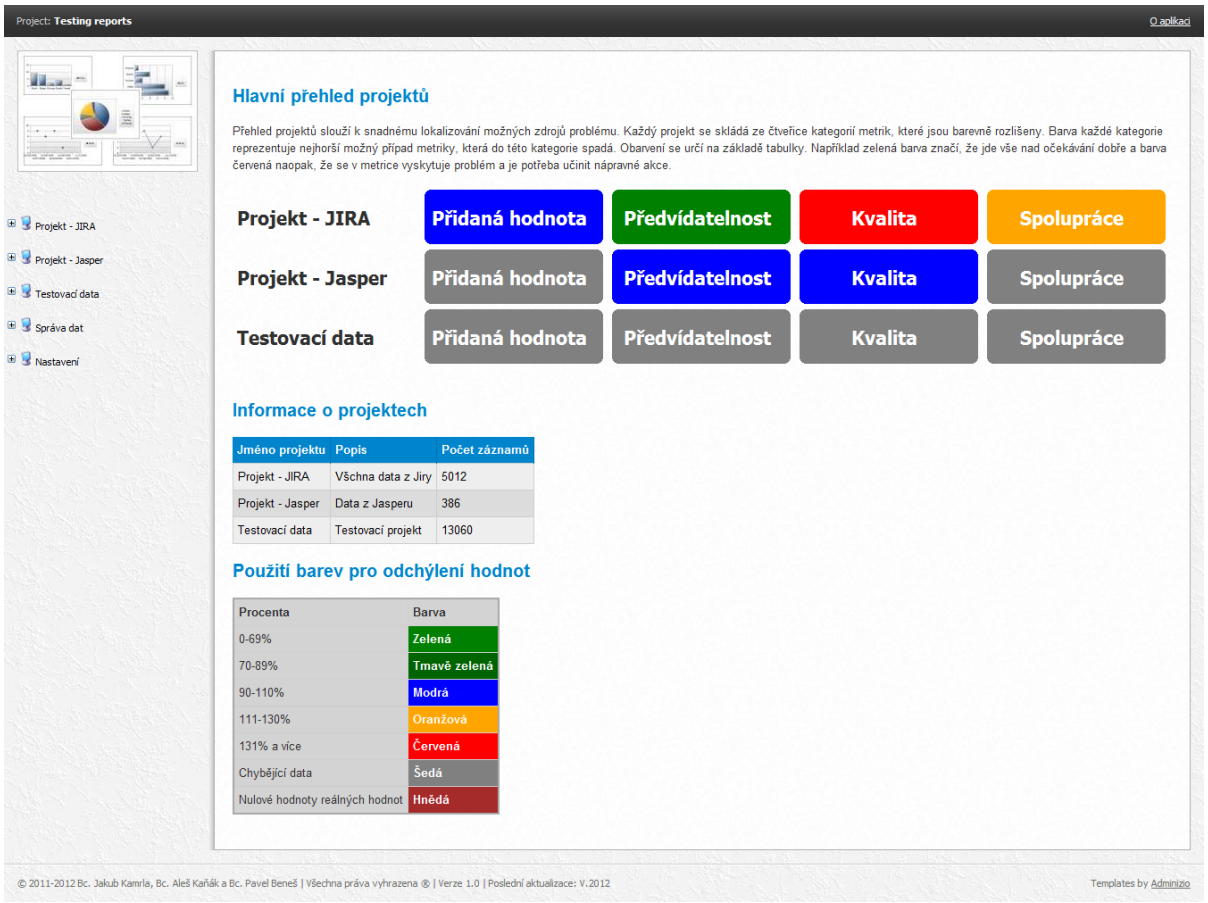
Přílohy

A. Obsah přiloženého DVD

/	*abstrakt.pdf*	abstrakt v českém a anglickém jazyce
/text		adresář s texty
	kan223-dp.pdf	text diplomové práce
	kam087-dp.pdf	text diplomové práce Bc. Jakuba Kamrly
/web_app/sql		adresář obsahující soubory s SQL příkazy pro vytvoření databáze a testovacích dat.
/web_app/srv		adresář obsahující verzi webové aplikace bez zdrojových kódů.
/web_app/src		adresář obsahující verzi webové aplikace včetně zdrojových kódů.

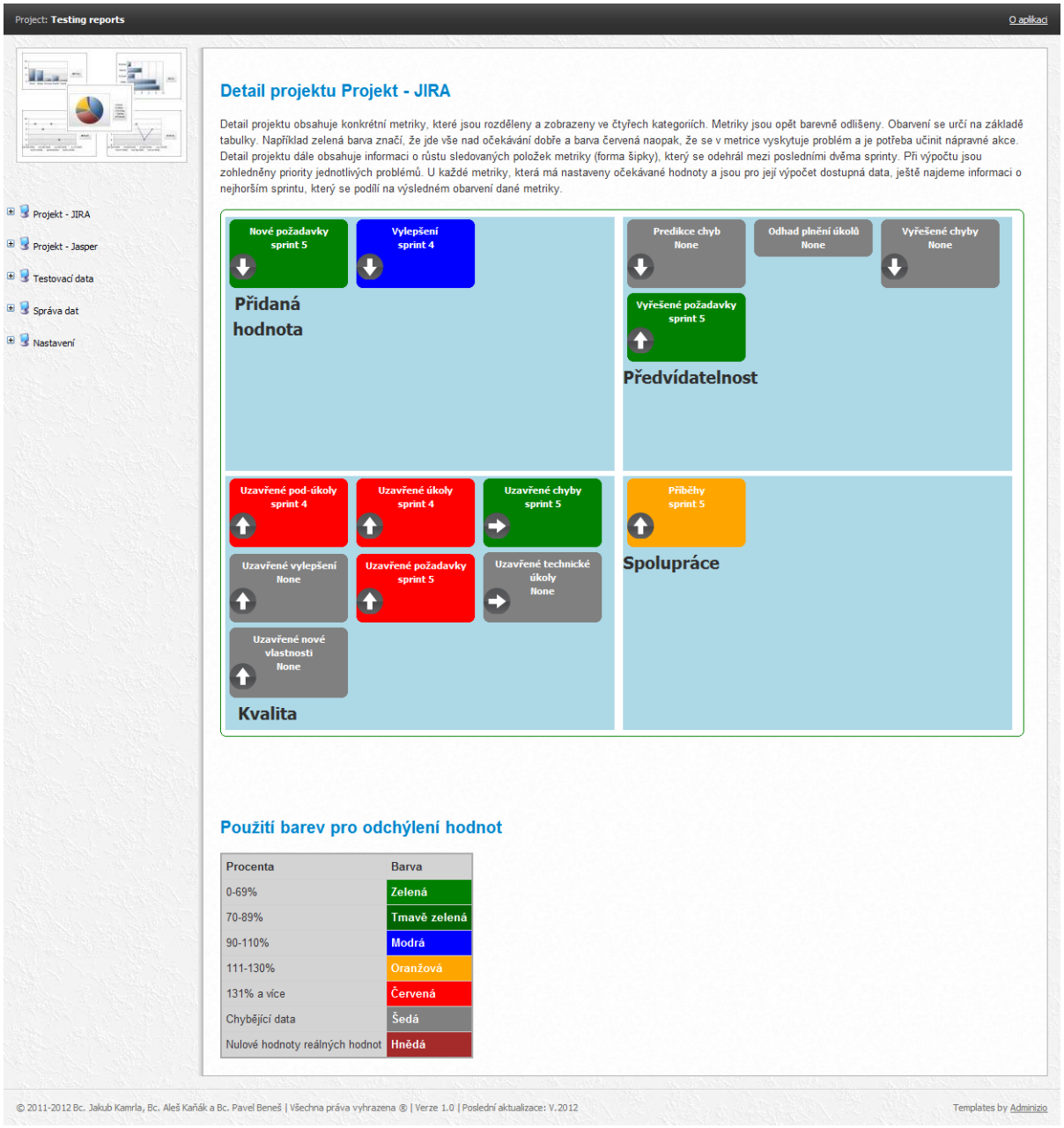
B. Ukázky uživatelského rozhraní

Na následujícím snímku (obrázek 35) můžeme vidět úvodní stránku webové aplikace *Testing reports*. Hlavní obsah stránky je vyplněn přehledem všech projektů v systému včetně hodnocení každé kategorie metrik s barevným odlišením. Vidíme zde i aktuální počet záznamů z každého projektu. V levém postranním panelu se kromě loga nachází také navigační menu.



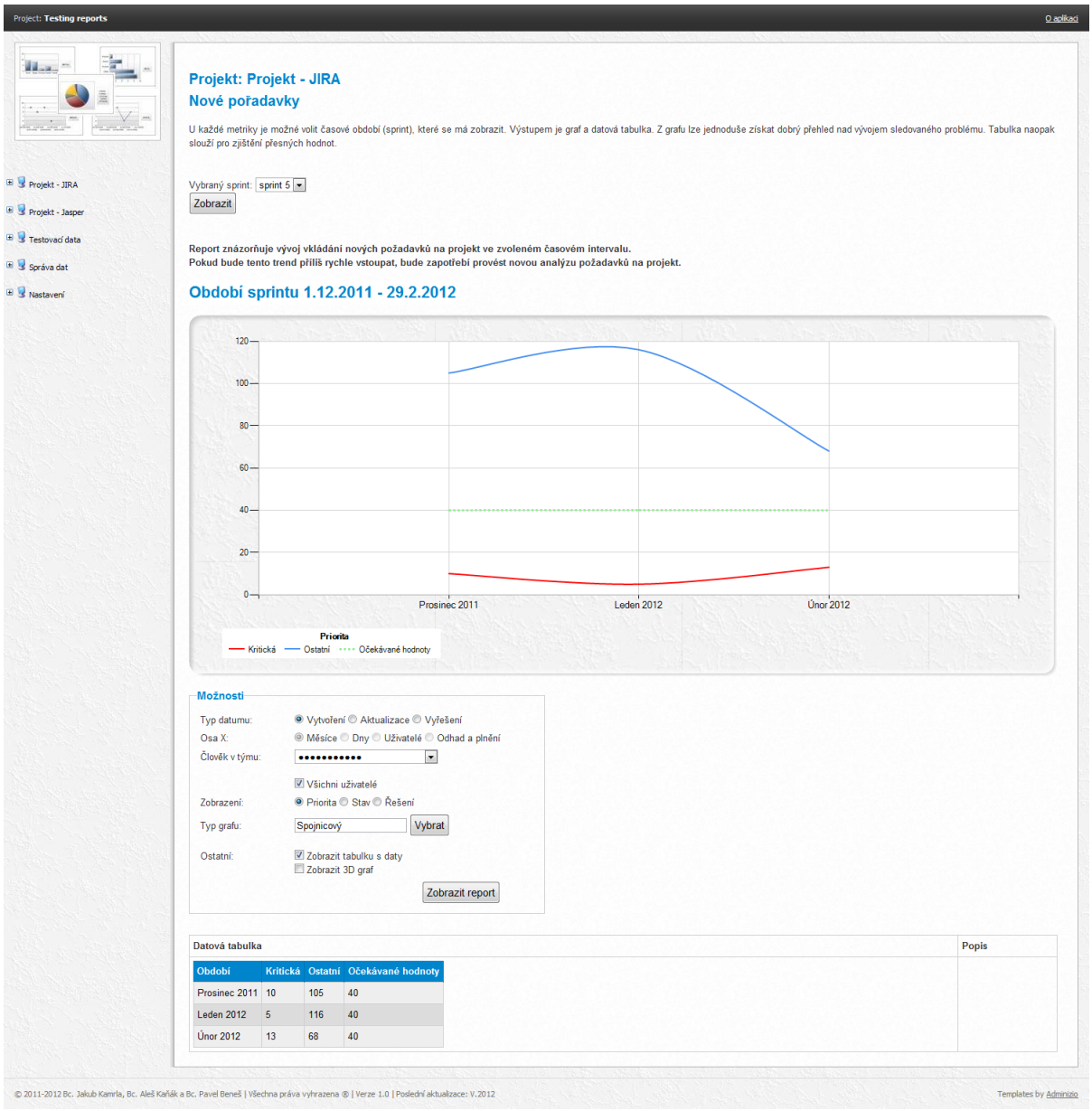
Obrázek 35: Přehled projektů

Detail projektu je zachycen na dalším snímku (obrázek 36). Jsou zde zobrazeny jednotlivé metriky vztahující se na daný projekt včetně jejich kategorií. Každá metrika nese barevnou informaci o jejím plnění (nastaveno dle tabulky). Dále u každé z metrik najdeme šipku, která indikuje růst položek problémů.



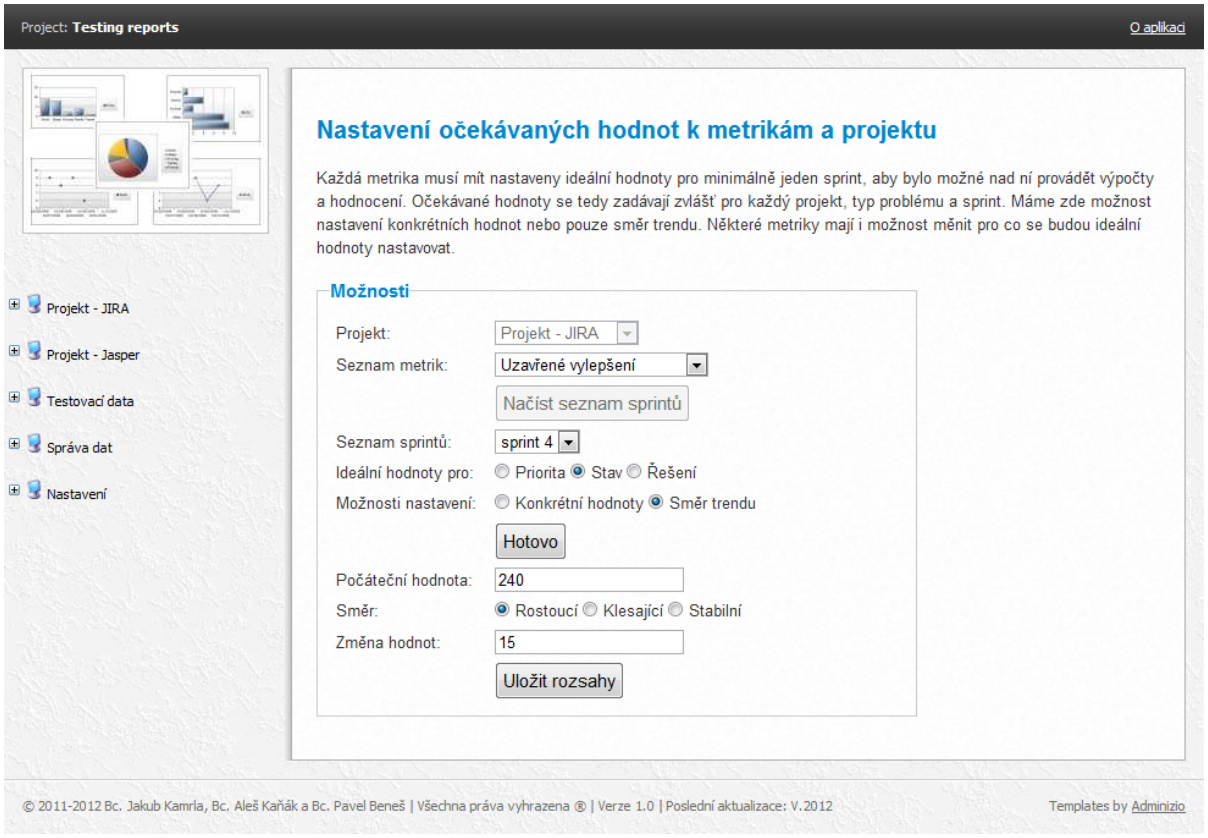
Obrázek 36: Detail projektu

Detail metriky (obrázek 37) obsahuje dílčí hodnoty a výsledky, které jsou znázorněny pro lepší přehlednost v grafu. K dispozici je i tabulka pro porovnání hodnot. Dále je možné měnit časové období (sprint) k zobrazení nebo typ vykresleného grafu díky filtru.



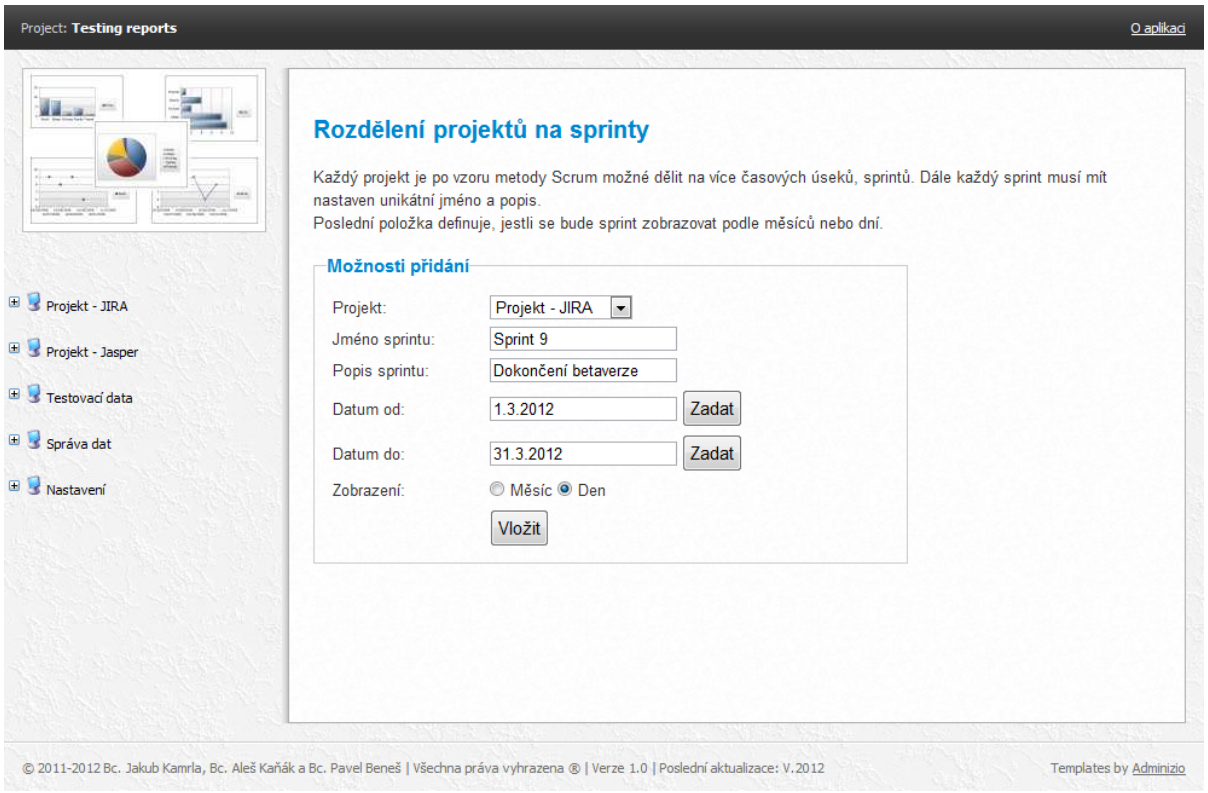
Obrázek 37: Detail metriky

Na následujícím snímku (obrázek 38) se nachází nastavení očekávaných hodnot pro libovolnou metriku ve vybraném projektu. U očekávaných hodnot lze volit mezi trendem nebo nastavením konkrétních hodnot. Lze také vybrat, ke kterému sprintu se očekávané hodnoty váží a na základě jakého kritéria se nastavují (priorita, stav, řešení).



Obrázek 38: Nastavení očekávaných hodnot

Rozdělení projektů na sprinty je zachyceno na posledním snímku (obrázek 39). Pro vybraný projekt zde lze vytvořit sprint včetně jeho pojmenování a popisu. Musíme zvolit celkové trvání sprintu (datum od/do) a také, jak se bude sprint zobrazovat v grafech (měsíc nebo den).



Obrázek 39: Rozdělení projektu na sprinty